(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 May 2001 (03.05.2001)

PCT

(10) International Publication Number
WO 01/31482 A2

(51) International Patent Classification[7]: G06F 17/00

(21) International Application Number: PCT/US00/29146

(22) International Filing Date: 20 October 2000 (20.10.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/425,579 22 October 1999 (22.10.1999) US

(71) Applicant: GENERAL ELECTRIC CAPITAL CORPORATION [US/US]; 44 Old Ridgebury Road, Danbury, CT 06810-5105 (US).

(72) Inventors: ARDITTI, Mark, Lee; 115 Walnut Grove Road, Ridgefield, CT 06877 (US). WILSON, Judith, A.; 20 Plumwood Road, Briarcliff Manor, NY 10510 (US). FAIELLA, Steven; 55 Mill Plain Road, Unit 22-4, Danbury, CT 06811 (US). PRICE, David, K.; 151 Lazy Brook Road, Monroe, CT 06468 (US).

(74) Agents: CHASKIN, Jay, L. et al.; General Electric Company, 3135 Easton Turnpike W3C, Fairfield, CT 06431 (US).

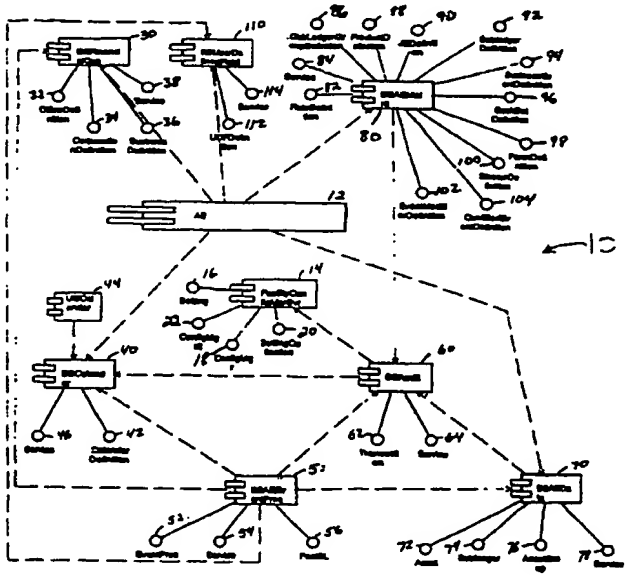
(81) Designated States *(national)*: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *Without international search report and to be republished upon receipt of that report.*

[Continued on next page]

(54) Title: LEASE AND LOAN SUB-LEDGER ACCOUNTING METHODS AND SYSTEM



(57) Abstract: A lease and loan sub-ledger accounting system (10) that provides sub-ledger transaction detail for asset level accounting is described. The accounting system includes a lease and loan accounting engine (12), a plurality of component object model (COM™) enabled sub-ledger accounting components, and a plurality of programmatic interfaces (140) enabling communication between the accounting components and the accounting engine.

WO 01/31482 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

LEASE AND LOAN SUB-LEDGER ACCOUNTING METHODS AND SYSTEM

BACKGROUND OF THE INVENTION

This invention relates generally to accounting systems and more specifically to accounting systems that provide support for leases and/or loans.

Typical lease and loan accounting systems include accounting support at a lease or loan level, as part of an integrated system. Known lease and loan accounting systems do not utilize a sub-ledger transaction to support transactions that comprise a general ledger, but instead provide sub-ledger transaction support indirectly from an operational system.

As a result accounting is not isolated from changes that occur in the operational system that should not have accounting impact. In addition, asset level detail typically required for complex lease and loan transactions may not be provided by such accounting systems.

BRIEF SUMMARY OF THE INVENTION

In one aspect, the present invention is an accounting system that supports multiple pricing models and supports multiple operational systems. In addition, the accounting system is isolated from operational system changes to provide stability to other accounting systems used simultaneously in background.

In an exemplary embodiment, a lease and loan sub-ledger accounting system for providing sub-ledger transaction detail for asset level accounting includes

programmatic interfaces for enabling communication with component object model (COM™) (COM is a trademark of the Microsoft Corporation) enabled lease or loan accounting systems.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates an architecture of an exemplary lease and loan sub-ledger accounting engine.

Figure 2 illustrates multiple document interfaces available to an administrator of a lease and loan sub-ledger accounting engine.

DETAILED DESCRIPTION OF THE INVENTION

Figure 1 illustrates an architecture of an exemplary lease and loan sub-ledger accounting system 10. A lease and loan sub-ledger accounting engine 12 distributes and receives formatted data directly or indirectly via a set of clearly defined program interfaces. Program interfaces enable any COM™ enabled lease or loan accounting operational system (not shown) to communicate with lease and loan sub-ledger accounting system 10, thereby isolating accounting functions from the operational system and providing sub-ledger transaction detail.

To provide control of the overall facility configuration and allow use of accounting system 10 to support multiple pricing models and multiple operational systems, accounting system 10 includes a facility configuration manager server 14 which includes classes of setting 16, configuration manager 18, setting collection 20, and second configuration manager 22.

Lease and loan sub-ledger accounting system 10 includes a financial organization package 30 that provides internal and external references to financial entities. Included in financial organization package 30 is an office definition class 32 containing methods defining an office to accounting engine 12. Financial organization package 30 also includes a corporation definition class 34 that contains methods defining a corporation to accounting engine 12. Financial organization

package 30 further includes a business definition class 36 containing methods defining a business to accounting engine 12. Financial organization package 30 also includes a service class 38, which provides service to financial organization package 30 by retrieving data organization data from accounting engine 12.

Financial organization package 30 and classes described above as well as package and class definitions that follow are described in further technical detail in Appendix A titled Accounting Engine Package Documentation. The descriptions set forth in Appendix B are descriptions of the various accounting functions contained within accounting engine 12.

Lease and loan sub-ledger accounting system 10 also includes a calendar package 40 to provide support for multiple fiscal calendars. Calendar package 40 includes a calendar definition class 42 used to identify a fiscal closing date for a bookset and resolve key activity dates used for periodic processes such as bank holidays. If an asset uses multiple booksets, those booksets all use the same calendar. A user service calendar package 44 is also included in calendar package 40 to allow calendar package 40 to run without a complete install of accounting engine 12. A service class 46 which provides service to calendar package 40 is also included.

Lease and loan sub-ledger accounting system 10 also includes an event processor package 50 to recognize a financial asset such as a piece of equipment, a lease, or a loan to also support account level or asset level accounting. Event processor package 50 includes an event processor class 52 containing methods used to interface with accounting engine 12 that require creation of journal entries and that are fundamental to transaction processing between the operational system and accounting engine 12. A service class 54 is included in event processor package 50 that contains encapsulated retrieval methods for event processor 50. Event processor 50 further contains a post sub-ledger class 56, which is a controller class used to create or modify sub-ledgers and their supporting transaction detail.

Lease and loan sub-ledger accounting system 10 also includes an audit package 60 that clearly identifies every transaction in accounting system 10 and allows the operational system to relate every accounting transaction with a corresponding operational transaction. Audit package 60 contains a transaction class 62 that contains methods to create and use a unique transaction identifier, which is recorded on all accounting engine 12 entities. A service class 64 is included in audit package 60 and is used to retrieve data.

A data package 70 is further included in lease and loan sub-ledger accounting system 10. Data package 70 includes an asset class 72 that represents a physical piece of equipment or a financial entity such as loan or an unapplied cash account. Data package 70 further includes a sub-ledger class 74 that performs additions and updates to the sub-ledger balance and detail for a single asset by ensuring debits and credits are written in matched pairs when posted. An asset group class 76 is included in data package 70 that provides a user definable financial asset grouping mechanism to accounting engine 12 to allow easy summarization by vendor, customer, branch, or office. Data package 70 further includes a service class 78 which acts as the service component of accounting engine 12 and services assets, sub-ledgers, event processor 50, and data streams.

Lease and loan sub-ledger accounting system 10 also includes a maintenance package 80. Maintenance package 80 includes a rule definition class 82 that contains methods for creating, using, and updating a rule in accounting engine 12. Maintenance package 80 also includes a service class 84 containing methods for servicing maintenance package 80. Maintenance package 80 further includes a sub-ledger group definition class 86 that defines sub-ledger groups to accounting engine 12. A product definition class 88 tailored to specific lease and loan accounting rules by containing methods for creating, using, and updating a product in accounting engine 12 is included in maintenance package 80.

A journal entry definition class 90 used to specify different debits and credits contains methods for creating or updating a journal entry in accounting engine 12 is included in maintenance package 80. Also included in maintenance package 80

is a sub-ledger definition class 92 containing methods for creating, updating, and using a sub-ledger chart of accounts in accounting engine 12. Maintenance package 80 further includes a business event definition class 94 that contains methods for creating, updating, and using a business event in accounting engine 12. A book set definition class 96 in maintenance package 80 contains methods for creating, updating, and using a book set in accounting engine 12 which enables accounting system 10 to use multiple types of generally accepted accounting principles.

Maintenance package 80 still further includes a parameter definition class 98 containing methods for creating, updating, and using a parameter in accounting engine 12. A stream definition class 100 in maintenance package 80 contains methods for creating, updating, and using data streams to compress the high volume of information for supporting asset level accounting and reducing storage requirements in accounting engine 12. An event modifier definition class 102 in maintenance package 80 contains methods for creating, updating, and getting an event modifier such as country, business, or product specific exceptions to an accounting event in accounting engine 12. A qualified event definition class 104 in maintenance package 80 is used to describe specific event combinations based on a financial product by creating product and business event association in accounting engine 12 using journal entries and event modifiers.

Qualified event definition class 104 of maintenance package 80 together with event processor package 50 provide a flexible event driven process model to allow accounting engine 12 to derive the correct accounting entry for a lease or loan accounting event.

In addition, maintenance package 80 and event processor package 50 provide user defined finance rules for determining a correct type of accounting entry based on existing information and calculation rules to support financial calculations needed to properly account for leases and loans in multiple business organizations and countries.

User defined field package 110 includes a user defined field definition class 112 and a service class 114 that provide capability and the services to define and add information needed to support specific accounting requirements.

Also included in accounting system 10, but not shown in Figure 1, are a currency package and an import/export package. Currency package (not shown) includes a currency definition class containing methods to create, update and use a currency in accounting system 10 and further contain currency rounding rules and a currency rate table thus providing multi-national detail in accounting system 10. Currency package also includes a service class that provides services for the currency package.

Import/export package (not shown) includes an import/export class with methods used for input/output operations of large amounts of data stored in file form.

Figure 2 is a flow diagram showing multiple document programmatic interfaces 140 available to an administrator to define how an accounting application is described to lease and loan sub-ledger accounting engine 12 (shown in Figure 1). Main interface 150 is a user interface that gives an administrator access to form interfaces. The definitions of the accounting application contained within the form interfaces are sent to the main executable module 152. The form interfaces are defined below. Most form interfaces have a plurality of operations available to an administrator. Form interfaces are listed below and are described in technical detail in Appendix C which is titled Form Interface Definitions.

Examples of form interfaces are: Sub-ledger Balances Report 154, Bookset 156, Product 158, User Defined Field Maintenance 160 and Calendar 162. Calendar 162 allows access to other form interfaces such as Current Fiscal Period 164, Calendar Activity Type 166, and Fiscal Period Start Dates 168. Calendar 162 and the form interfaces which Calendar 162 allows access to are used to view, select, and maintain calendars and fiscal periods and to view, add, and delete activity types.

Other examples of form interfaces are: Sub-ledger Chart Groups 170, used to add, update, delete, and display sub-ledger groups. Qualified Event Inquiry 172, Journal Entry Maintenance 174, used to maintain journal entry headers, Event Modifier Maintenance 176, Organization Maintenance 178, and Sub-ledger Chart of Accounts 180 used to add, update, and delete subledger chart of accounts. Organization Maintenance 178 form interface allows access to other form interfaces such as Office Maintenance 182 and Business Maintenance 184. Office Maintenance 182 form interface further allows access to Office Maintenance Part Two 186 and Business Maintenance 184 form interface allows access to Business Add 188 form interface.

Other form interfaces shown in Figure 2 are Rule Maintenance 190 and Qualified Event Maintenance 192. Rule Maintenance 190 form interface allows access to form interface Rule Maintenance Lines 194 which in turn allows access to form interface Parameter Maintenance 196. Qualified Event Maintenance 192 form interface allows access to other form interfaces such as Qualified Event Lines 198 and Product Pick 200. Qualified Event Lines 198 allows access to form interface Qualified Event Parameters Maintenance 202. Qualified Event Parameters Maintenance 202 also allows access to form interface Parameter Maintenance 196.

Lease and loan sub-ledger accounting system 10 is capable of supporting multiple pricing models and multiple operational systems. That capability provides stability when used with the accounting system of choice by isolating accounting engine 12 from the operational system. Therefore, the ability to change operational systems without negatively impacting the accounting system is enhanced. In addition, asset level detail is provided that is required for complex lease and loan transactions.

While the invention has been described in terms of various specific embodiments, those skilled in the art will recognize that the invention can be practiced with modification within the spirit scope of the claims.

BSAE Data Package Overview

Description

Classes

IAsset
IAssetGroup
IService
ISubledger

Subpackages

None

IAsset Class

Description

The Asset can represent a physical piece of equipment or a financial entity such as a loan or an unapplied cash account. All Assets will have a corresponding Asset represented on the source (ATLAS) system.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

Long Create(ADOR.Recordset byval arsAsset, Long alTransId)

Class:	IAsset
Description:	This will create one asset using the recordset. This operation will be invoked after IService.GetAsset (0) has been used to return an empty recordset which can be populated with valid asset data by the operational (ATLAS) system. This will return the asset entity id for the asset created.
Inputs:	byval arsAsset - alTransId -
Outputs:	None
Returns:	Long

Long Update(ADOR.Recordset byval arsAsset, Long alTransId)

Class:	IAsset
Description:	Modify an asset using the ADOR.Recordset.
Inputs:	byval arsAsset - alTransId -
Outputs:	None
Returns:	Long

String Ping()

Class:	IAsset
Description:	

	Return a string indicating whether this object is instantiated.
Inputs:	**None**
Outputs:	**None**
Returns:	**String**

byval astrExtAssetGroupRef -
byval avAssetEntityIdst -
byref TransId -

Outputs: None
Returns: long

Boolean Delete(long alEntityId)

Class: IAssetGroup
Description:

This will remove an Asset Group (not the individual assets) from the AE.

Inputs: alEntityId -
Outputs: None
Returns: Boolean

short RemoveAssets(Long alEnitityId, VariantArray avAssetEntityIDs)

Class: IAssetGroup
Description:

Remove one or more Assets from an Asset Group using the list of assets specified in the array. If 'ALL' is specified then all Assets will be disassociated with this Asset Group.
Return a count of the assets removed from the Asset Group.

Inputs: alEnitityId -
avAssetEntityIDs -
Outputs: None
Returns: short

String Ping()

Class: IAssetGroup
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

IAssetGroup Class

Description

The accounting engine does accounting at the asset level. Often, an operational system will be set up to perform some activities at a higher level; e.g., customer or account level. The asset group record allows any number of assets to be grouped together based upon the group name. It will be possible for the operational system to pass an Asset Group, rather than an array of individual assets, to the Accounting Engine. Using the Asset Group will cause all assets associated with the Group to be processed through an Event. Assets group types may be on a customer or account level.

There are two distinct advantages to using an Asset Group rather than passing an array of assets:
1. The Asset Group will have an easily recognizable value in the operational system; e.g., Asset 1, 3, and 17 may all belong to Customer ABC.
2. It will minimize the amount of data passed for frequently-referenced Asset Groups or Asset Groups with many assets.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

short AddAssets(Long byval alEntityId, VariantArray byval avAssetEntityIds)

Class:	IAssetGroup
Description:	Add one or more assets to an Asset Group using Asset entity id 's passed in the array. Reutrn a count of assets added to the group..
Inputs:	byval alEntityId - byval avAssetEntityIds -
Outputs:	None
Returns:	short

long Create(Long byval alFacilityId, string byval astrExtAssetGroupType, string byval astrExtAssetGroupRef, VariantArrayal byval avAssetEntityIdst, long byref TransId)

Class:	IAssetGroup
Description:	Create one Asset Group and associate existing assets with the group. If the Asset Group already exists this will raise an error. Return the entity id of the asset group created.
Inputs:	byval alFacilityId - byval astrExtAssetGroupType -

IService Class

Description

This is the service component of the Accounting Engine. This will service: Assets, Subledgers, The Event Processor, and streams.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

String Ping()

Class:	IService
Description:	Return a string indicating whether this object is instantiated.
Inputs:	None
Outputs:	None
Returns:	String

ADOR.Recordset GetAllAssetBooksetsByID(long byval alEntityID)

Class:	IService
Description:	Get all of the Booksets associated with an asset by the asset id. Each asset can be used to make entries in multiple booksets.
Inputs:	byval alEntityID -
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetAllAssetGroupTypes()

Class:	IService
Description:	Get all of the Asset Group Types in the AE database. Group types are used to identify / stratify the asset groups that have been created. e.g. ATLAS may create a loan Asset Group and a Customer Asset Group. Each of these may have the EntityId 1234567 in ATLAS, since they represent different data. The Ae needs to know what kind of group type (Customer or Loan) to retrieve if asset group value 1234567 is specified.
Inputs:	None
Outputs:	None

Returns: ADOR.Recordset

ADOR.Recordset GetAllAssetProductsByID(long byval alEntityID)

Class: IService
Description:

Get a list of all of the products that are associated with this asset. An asset may behave like a tax product in one set of books and a loan product in another set of books.

Inputs: byval alEntityID -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetAllAssetTypes()

Class: IService
Description:

This is used to return all of the asset types in the AE. This is used to subclass assets. Is this asset a loan, a piece of equipment or a suspense account?

Inputs: None
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetAllOfficeCorps()

Class: IService
Description:

Get all of the Office Corps in the AE. This is the junction of valid office / corp combinations.

Inputs: None
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetAllSLBalancesByAssetId(long byval alAssetEntityId, byval astrYear as string, BSAEDataISvcPeriodEnum byval aPeriod)

Class: IService
Description:

This will return a series of Subledger balances for a single asset and a single period.
This needs to include the Subledger name, EntityID and amount for every SL found for the asset.

Inputs: byval alAssetEntityId -
byval astrYear as string -
byval aPeriod -
Outputs: None

Returns: ADOR.Recordset

ADOR.Recordset GetAsset(Long byval alAssetId)

Class: IService
Description:

This is used to return asset data in a recordset.
Use this operation before creating an asset to return an empty recordset set by specifying asset 0 in the arguement. Specify a valid asset id to return asset data.

Inputs: byval alAssetId -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetAssetGroupAssets(long byval alEntityID)

Class: IService
Description:

Getlist of all the assets in a single asset group and return the details in ADOR.Recordset.

Inputs: byval alEntityID -
Outputs: None
Returns: ADOR.Recordset

ADOR.Records GetSLBalanceForAssetByID(long byval alAssetEntityId, long byval alCOAEntityid, byval astrYear as string, BSAEDataISvcPeriodEnum byval aPeriod)

Class: IService
Description:

This will return a single Subledger balance for an asset.

Inputs: byval alAssetEntityId -
byval alCOAEntityid -
byval astrYear as string -
byval aPeriod -
Outputs: None
Returns: ADOR.Records

ADOR.Recordset GetSLBalanceForAssetGroupByID(long byval alCOAEntityid, long byval alGroupId, byval astrYear as string, BSAEDataISvcPeriodEnum byval aPeriod)

Class: IService
Description:

Get a subledger balance for an asset group.

Inputs: byval alCOAEntityid -
byval alGroupId -
byval astrYear as string -
byval aPeriod -
Outputs: None

Returns: ADOR.Recordset

ADOR.Recordset GetSLDetailByAssetGroupID(long byval alAssetGroupId, long byval alCOAEntityId, date byval adteFrom, date byval adteTo)

Class: IService
Description:
This will return all of the Subledger details found for a subledger for the specified asset and date range. Include rows matching the from and to date in the result set. Include subledger header information
Inputs: byval alAssetGroupId -
byval alCOAEntityId -
byval adteFrom -
byval adteTo -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetSLDetailBySLandAsset(long byval AssetEntityId, long byval alCOAEntityId, date byval adteFrom, date byval adteTo)

Class: IService
Description:
This will return the Subledger details found for a single subledger for the specified asset and date range. Include rows matching the from and to date in the result set.
Incidue the subledger header information.
Inputs: byval AssetEntityId -
byval alCOAEntityId -
byval adteFrom -
byval adteTo -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetSLDetailBySLGroupAsset(long byval alAssetID, long byval alSLGroupID, date byval adteFrom, Date byval adteTo)

Class: IService
Description:
Get the Subledger detail for a subledger group associated with a single asset.
Inputs: byval alAssetID -
byval alSLGroupID -
byval adteFrom -
byval adteTo -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetSLGroupBalanceForAssetByID(long byval alAssetEntityId, long byval alCOAEntityId, byval astrYear as string, BSAEDataISvcPeriodEnum byval aPeriod)

Class: IService
Description:

This will return the sum of the balances for the Subledgers in a Subledger Group for the requested asset.

Inputs: byval alAssetEntityId -
byval alCOAEntityId -
byval astrYear as string -
byval aPeriod -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetSLGroupBalanceForAssetGroupByIDs(long byval alSLGroupID, long byval alAssetGroupID, string byval astrYear, BSAEDataISvcPeriodEnum byval aPeriod)

Class: IService
Description:

Get the sum of the balances for a single subledger group, for an entire asset group.

Inputs: byval alSLGroupID -
byval alAssetGroupID -
byval astrYear -
byval aPeriod -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetSLGroupYearByAssetID(long byval alAsset, long byval alSLGroupID, string byval astrYear)

Class: IService
Description:

Get the subledgers balances for an entire subledger group for a single year for a single asset.

Inputs: byval alAsset -
byval alSLGroupID -
byval astrYear -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetSLYearByAssetID(long byval alAssetId, long byval alCOAEntityID, string byval astrYear)

Class: IService
Description:

This will return all of the balances found on a single subledger account for the year and asset passed in to this method.

Inputs:	byval alAssetId - byval alCOAEntityID - byval astrYear -
Outputs:	None
Returns:	ADOR.Recordset

ISubledger Class

Description

This is used to perform additions and updates to the SL Balance and Detail for a single asset. We will ensure that Debits and Credits are written in matching pairs by processing the Debit and Credit using a single invocation of the ISubledger Post method to create both sides of the SL entry.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

String Ping()

Class:	ISubledger
Description:	Return a string indicating whether this object is instantiated.
Inputs:	None
Outputs:	None
Returns:	String

Rollover()

Class:	ISubledger
Description:	This is the method user for year end processing. It will be necessary to close the old year and start and new year. This is considered a S/L account rollover. At the end of a Fiscal year the 12/31 balances are finalized and 1/1 balances are created for the new year. There are many valid reasons the 12/31 balance does not need to = the 1/1 balance. The 1/1 balance may be zero or it may be the total of several other Subledgers that have been rolled in the new 1/1 balance.
Inputs:	None
Outputs:	None
Returns:	None

Long Post(long byval alEntityId, variantarray byval avarPostData)

Class:	ISubledger
Description:	

Create or update the Subledger balance and create a corresponding Subledger detail. This is an all or nothing unit of work.

The variant array contains all of the data needed to post one or more debit / credit pairs. It will always work on at least one debit and one credit.

Rules:
1. If the SL_Balance does not exist then invoke create to create the SL for this asset and the invoke CreateYear to create a new year of SL_balance for this asset.
2. Fiscal period needs to be resolved using the effective date. All posting will occur in the current fiscal period for this calendar.
3. Post the balance to the fiscal month. The S/L balance needs to be propogated forward from the transaction date for all months in the transaction year.
4. Invoke CreateDetail to create the SL_Detail row.
5. abooReverseOperator needs to be inspected to determine how acurAmount should be signed. If abooReverseOperator = true then acurAmount should be reversed by multiplying by -1.
6. For credits subtract the amount being posted, for debits add the amount. Since Post is calculating the correct operator, pass the correct signed amount to CreateDetail
7. Return the alTransId created by the Audit component as long.

Inputs: byval alEntityId -
byval avarPostData -
Outputs: None
Returns: Long

PrivateAccess Methods

Long CreateDetail(long byval alSLBalanceEntityId, date byval adteEffective, currency byval acurAmount, string byval astrDebitCredit, long byval alBankEntityId, long byval alJEEntityId, date byval adtePeriod)

Class: lSubledger
Description:

This is the only method used to create the supporting detail for the sl balance. This is an important Audit point.
This will be invoked by Post, Rollover.
All fields are required except for Bank.
Return the EntityID of the debit or credit created.

Inputs: byval alSLBalanceEntityId -
byval adteEffective -
byval acurAmount -
byval astrDebitCredit -
byval alBankEntityId -
byval alJEEntityId -
byval adtePeriod -
Outputs: None
Returns: Long

Long Create(long byval alAssetEntityId, long alCOAEntityId, date byval adteEffective, currency byval acurAmount, string byval astrDebitCredit, long byval alProductEntityId, long byval alBankEntityId, long byval alJEEntityId, long byval alProductEntityId, long alCorpEntityId, long byval alOfficeEntityId, long alBusinessEntityId, Boolean byval abooReverse~perator)

Class: ISubledger
Description:

1. Create one row in the SL_Balance table.
2. This is invoked from the Post or Rollover methods when the Subledger Balance does not already exist for the posting.
3. All fields on the SL_Balance table are required.

Inputs: byval alAssetEntityId -
alCOAEntityId -
byval adteEffective -
byval acurAmount -
byval astrDebitCredit -
byval alProductEntityId -
byval alBankEntityId -
byval alJEEntityId -
byval alProductEntityId -
alCorpEntityId -
byval alOfficeEntityId -
alBusinessEntityId -
byval abooReverseOperator -
Outputs: None
Returns: Long

long CreateYear(Long byval alSLBalanceEntityId, date byval adteYear)

Class: ISubledger
Description:

1. This will add a row to the SL_Monthly_Balances table for the year specified.
2. This can only be the current year or next year.
3. All balances will be initialized to zero.
4. The year will be passed in from the method that invoked CreateYear.

6, Return alEntityId as long.

Inputs:	byval alSLBalanceEntityId - byval adteYear -
Outputs:	None
Returns:	long

BSAEMaint Package Overview

Description

This package contains the business service classes required to support the user maintenance of Accounting Engine data.

Classes

IBooksetDefinition
IBusinessEventDefinition
IEventModifierDefinition
IJEDefinition
IParmDefinition
IProductDefinition
IQualifiedEventDefinition
IRuleDefinition
IService
IStreamDefinition
ISubLedgerGroupDefinition
ISubledgerDefintion

Subpackages

None

IBooksetDefinition Class

Description

This interface contains the methods required to create, update and use a Bookset in the AE. This will maintain the Bookset (lookup table) entity: Bookset name and description.

alTaxTytpeID: Id to table indentifying Tax, Book, Both
alReportTypeId: Id to table identifying Local, U.S. or Both

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

Long Create(String byval astrBookSetName, string byval astrDesc, Integer byval alActive, long byval alTaxTypeId, long byval alReportTypeId, Long byref alTransNbr)

Class: IBooksetDefinition
Description:

This will create a Bookset in the Accounting Engine. Before any Asset or Event can refer to a Bookset, it will be necessary to define (create) the Bookset entity.

- Error handler will handle duplicate BooksetName error and foreign key errors.
- Get todays date for adteStatusDate.
- Insert into Bookset
- alTransNbr = Call LogTransId
- Return ID as long

Inputs: byval astrBookSetName -
byval astrDesc -
byval alActive -
byval alTaxTypeId -
byval alReportTypeId -
byref alTransNbr -
Outputs: None
Returns: Long

Delete(Long byval alEntityId, Long byref alTransNbr)

Class: IBooksetDefinition
Description:

This will delete a Bookset in the Accounting Engine.

· Delete from Bookset where SQ_BOOKSET_ID = alEntityId.
· alTransNbr = Call LogTransId.

Inputs: byval alEntityId -
byref alTransNbr -
Outputs: None
Returns: None

Update(Long byval alEntityId, string byval astrBooksetName, string byval astrBooksetDesc, integer byval alActiveId, Long byval alTaxTypeId, long byval alReportTypeId, Long byref alTransNbr, string byval astrDescription)

Class: IBooksetDefinition
Description:

This will update one Bookset in the Accounting Engine.

· Get todays date for adteStatusDate
· Update Bookset

Inputs: byval alEntityId -
byval astrBooksetName -
byval astrBooksetDesc -
byval alActiveId -
byval alTaxTypeId -
byval alReportTypeId -
byref alTransNbr -
byval astrDescription -
Outputs: None
Returns: None

String Ping()

Class: IBooksetDefinition
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

IBusinessEventDefinition Class

Description

This interface contains the methods required to create, update, and use a Business Event in the Accounting Engine. This will maintain the Event (Lookup table) entity: Business Event name, description.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

Long Create(String byval astrEventName, String byval astrEventDesc, Long byref alTransNbr)

Class: IBusinessEventDefinition

Description:

This will create a Business Event in the Accounting Engine. Before any Qualified Event can refer to a Business Event, it will be necessary to define (create) the Business Event entity. Return the Entity Id for the Business Event that has been created.

- Insert new Business_Event
- a.TransNbr = Call LogTransId
- Return ID as long

Inputs: byval astrEventName -
byval astrEventDesc -
byref alTransNbr -

Outputs: None

Returns: Long

Delete(Long byval alEntityId, Long byref alTransNbr)

Class: IBusinessEventDefinition

Description:

This will delete a Business Event in the Accounting Engine.

- Delete from Business_Event
- alTransNbr = Call LogTransId.

Inputs: byval alEntityId -

byref alTransNbr -
Outputs: None
Returns: None

Update(Long byval alEntityId, string byval astrEntityDesc, Long byref alTransNbr)
Class: IBusinessEventDefinition
Description:
This will update one Business Event in the Accounting Engine.

- Update Business_Event using astrEntityDesc

- alTransNbr = Call LogTransId.

Inputs: byval alEntityId -
byval astrEntityDesc -
byref alTransNbr -
Outputs: None
Returns: None

String Ping()
Class: IBusinessEventDefinition
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

IEventModifierDefinition Class

Description

This interface contains the methods required to create, update, and get an Event Modifier in the Accounting Engine.
The Event Modifier will be organized as a header and two unrelated sets of detail rows.
The header is used for the name and description of the modifer.
There is one collection of detail parameters that apply to this specific Event Modifer. This is information, in addittion to the standard parameter list required for the Business Event Product.
There is a another collection of details that are used to define the Event Modifier using Source, Field, Relational Operator and Value.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

Long Create(String byval astrEventModifierName, String byval astrEventModifierDesc, ADOR.Recordset byval arsLines, Long byref alTransNbr)

Class:	IEventModifierDefinition
Description:	This will create an Event Modifier in the Accounting Engine. Before an Event can refer to an Event Modifier it will be necessary to define (create) the Event Modifier entity, including any Event Modifier Lines that are required to define the logical Event Modifier. Create the Event Modifier Invoke private method UpdateRSLines Invoke private method UpdateRSParms, if any. Return the Entity Id of the Event Modifer that has been created.
Inputs:	byval astrEventModifierName - byval astrEventModifierDesc - byval arsLines - byref alTransNbr -
Outputs:	None
Returns:	Long

Delete(long byval alEntityId, ADOR.Recorset byval arsLines, long byref alTransNbr, ADOR.Recorset optional arsParms)

Class:	IEventModifierDefinition
Description:	

This will delete an Event Modifer, its associated Event Modifer Lines and the Event Modifier Parm List from the Accounting Engine.

Referential integrity will need to be enforced for the Qualified Event. It is only possible to delete an Event Modifier if there are no QE's that use it.

Inputs: byval alEntityId -
byval arsLines -
byref alTransNbr -
optional arsParms -
Outputs: None
Returns: None

String Ping()
Class: IEventModifierDefinition
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

Long Update(long byval alEntityId, string byval astrName, string byval astrDesc, ADOR.Recordset byval arsLines, long byref alTransId)
Class: IEventModifierDefinition
Description:
Update the name, description or Event Modifier Lines for this Event Modifier.
Inputs: byval alEntityId -
byval astrName -
byval astrDesc -
byval arsLines -
byref alTransId -
Outputs: None
Returns: Long

PrivateAccess Methods

UpdateEventModData(ADOR.Recorset byval arsLines)
Class: IEventModifierDefinition
Description:
Update the Event Modifer Lines or Parms using a Recordset.
Inputs: byval arsLines -
Outputs: None
Returns: None

IJEDefinition Class

Description

This interface contains the methods required to create or update a JE in the Accounting Engine. This will maintain the Journal Entry (Lookup table) entity: JE Name, description, (DR/CR pairs).

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

long Create(string byval astrName, string byval astrDesc, boolean byval abooManual, long byval alJENumber, ADOR.Recordset byval arsJEDetail, long byref alTransNbr)

Class:	**IJEDefinition**
Description:	
	This will define a Journal Entry to the Accounting Engine.
	The ADOR.Recordset contains the list of debit/credit pairs for this JE.
	1. Insert JE **2. Insert debit / credit pairs using ADOR.Recordset and the private method UpdateJEDetailRS.** **3. alTransNbr = Call LogTransId.** **4. Return ID as long**
Inputs:	**byval astrName -** **byval astrDesc -** **byval abooManual -** **byval alJENumber -** **byval arsJEDetail -** **byref alTransNbr -**
Outputs:	**None**
Returns:	**long**

Delete(string byval alEntityId, long byref alTransNbr)

Class:	**IJEDefinition**
Description:	

This will delete a Journal Entry in the Accounting Engine. Referential integrity needs to be enforced.

- Delete all the JE_DEBIT_CREDIT_PAIR for this alEntityId.
- Delete the JE for this alEntityId.
- alTransNbr = Call LogTransId.

Inputs: byval alEntityId -
byref alTransNbr -
Outputs: None
Returns: None

Update(long byval alEntityId, string byval astrDesc, boolean byval abooManual, long byval alJeNumber, ADOR.Recordset byval arsJEDetail, long byref alTransNbr)

Class: IJEDefinition
Description:

This will update one Journal Entry header in the Accounting Engine and the corresponding JE detail.

. Update the JE.
update the je detail using the ADOR.Recordset.
. return the LogTransId.

Inputs: byval alEntityId -
byval astrDesc -
byval abooManual -
byval alJeNumber -
byval arsJEDetail -
byref alTransNbr -
Outputs: None
Returns: None

String Ping()

Class: IJEDefinition
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

PrivateAccess Methods

UpdateJEDetailRS(ADOR.Recordset byval arsJEDetail, ObjectContext byval aobjContext, BSAEMaint.cDataClass byval aobjDataClass, BSAudit.ITransaction :byval alTransId : long byval aobjAudit)

Class: IJEDefinition
Description:

This will be used to add or remove debit / credit pairs from this JE.
It will always be necessary to get the ADOR.Recordset before using this method. Note: an empty recordset will be returned if there are no debit / credit pairs for this JE. This empty recordset can then be used in this method to insert debits and credits just as if this is an ordinary recordset update.

Inputs:	byval arsJEDetail - byval aobjContext - byval aobjDataClass - byval aobjAudit -
Outputs:	None
Returns:	None

IParmDefinition Class

Description

This interface contains the methods required to create, update, and use a Parameter in the Accounting Engine. This will maintain the Parameter entities: Parameter Name, Description and Parameter Type.
Parameter Type needs to be initially populated using SQL. This will not change often enough to write the definition methods to support this table. Parameter Type will contain values like: String, Numeric, Currency.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

long Create(string byval astrParmName, string byval astrParmDesc, string byval alEntityId, long byref alTransNbr, String byval astrParmTypeID)

Class:	IParmDefinition
Description:	This will create a Parameter in the Accounting Engine. Before a Product Business Event can refer to a Parameter it will be necessary to define (create) the Parm entity. - Insert into PARM - alTransNbr = Call LogTransId. - Return ID as long
Inputs:	byval astrParmName - byval astrParmDesc - byval alEntityId - byref alTransNbr - byval astrParmTypeID -
Outputs:	None
Returns:	long

Delete(long byval alEntityId, long byref alTransNbr)

Class:	IParmDefinition
Description:	This will delete an Parm in the Accounting Engine. - Delete from PARM - alTransNbr = Call LogTransId.

Inputs:	byval alEntityId - byref alTransNbr -
Outputs:	None
Returns:	None

Update(long byval alEntityId, string byval astrParmDesc, long byref alTransNbr, String byval astrParmTypeId)

Class:	IParmDefinition
Description:	This will update one Parm in the Accounting Engine. - Update PARM value(astrParmDesc) - alTransNbr = Call LogTransId.
Inputs:	byval alEntityId - byval astrParmDesc - byref alTransNbr - byval astrParmTypeId -
Outputs:	None
Returns:	None

String Ping()

Class:	IParmDefinition
Description:	Return a string indicating whether this object is instantiated.
Inputs:	None
Outputs:	None
Returns:	String

IProductDefinition Class

Description

This interface contains the methods required to create, update, and use a Product.in the Accounting Engine. This will maintain the Product (Lookup table) entity: Product name, description.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

long Create(String byval astrName, string byval astrDesc, long byref alTransNbr)

Class:	**IProductDefinition**
Description:	**This will create a Product in the Accounting Engine. Before any Asset can refer to a Product, it will be necessary to define (create) the Product entity.** **· Validate required fields: astrName and astrDesc** **· Check for duplicate on Product_Name** **· Insert into Product_AE** **· alTransNbr = Call LogTransId.** **· Return ID as long**
Inputs:	**byval astrName -** **byval astrDesc -** **byref alTransNbr -**
Outputs:	**None**
Returns:	**long**

Delete(Long byval alEntityId, long byref alTransNbr)

Class:	**IProductDefinition**
Description:	**This will delete a Product in the Accounting Engine. Referential integrity needs to be enforced.** **· Delete from Product_AE** **· alTransNbr = Call LogTransId.**
Inputs:	**byval alEntityId -** **byref alTransNbr -**
Outputs:	**None**
Returns:	**None**

Update(Long byval alEntityId, string byval astrDesc, Long byref alTransNbr)

Class:	IProductDefinition
Description:	This will update one Product in the Accounting Engine. Product name can not be changed · Validate required field: alEntityId. · Update Product_AE values(astrDesc) · alTransNbr = Call LogTransId.
Inputs:	byval alEntityId - byval astrDesc - byref alTransNbr -
Outputs:	None
Returns:	None

String Ping()

Class:	IProductDefinition
Description:	Return a string indicating whether this object is instantiated.
Inputs:	None
Outputs:	None
Returns:	String

IQualifiedEventDefinition Class

Description

This is where the pieces come together: product, business event, je, event modifier and Rules.There is no Update method for this interface. It will be necessary to Delete and Create a new Qualified Event.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

CreateProductBusinessEvent(long byval alBusinessEventEntityId, long byval alProductEntityId, long byref alTransNbr, ADOR.Recordset byval optional arsParms)

Class:	IQualifiedEventDefinition
Description:	This will create a Product and Business Event association in the Accounting Engine. Before any Qualified Event can refer to a Business Event, it will be necessary to define (create) the Business Event entity, the Product Entity and associate the Business Event and Product. Return the Entity Id for the Business Event Product that has been created. Insert Business Event / Product Insert Parms using ADOR.Recordset alTransNbr = Call LogTransId.
Inputs:	byval alBusinessEventEntityId - byval alProductEntityId - byref alTransNbr - byval optional arsParms -
Outputs:	None
Returns:	None

DeleteProductBusinessEvent(long byval alBusinessEventEntityId, long byval alProductEntityId, long byref alTransNbr)

Class:	IQualifiedEventDefinition
Description:	This will delete a Product and Business Event association from the Accounting Engine and the

association between parms and the Product Business Event..

. Delete the association between the parms and the Product Business Event.
. Delete from Product_Business_Event
. alTransNbr = Call LogTransId.

Inputs: byval alBusinessEventEntityId -
byval alProductEntityId -
byref alTransNbr -
Outputs: None
Returns: None

String Ping()
Class: IQualifiedEventDefinition
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

UpdateProductEventParms(string byval alBusinessEventEntityId, long byval alProductEntityId, byref alTransNbr as long, ADOR.Recordset byval arsParms)
Class: IQualifiedEventDefinition
Description:

This will update the Product Business Event association with Parameters in the Accounting Engine.
. update parms using Product_Business_Event_Parm using ADOR.Recordset
. Return TransId.

Inputs: byval alBusinessEventEntityId -
byval alProductEntityId -
byref alTransNbr as long -
byval arsParms -
Outputs: None
Returns: None

CreateQualEventLine(long byval alBusinessEventEntityId, long byval alProductId, long byval alEventModID, long alRuleId, long byval alJEID, longastrEntryName : string byval alJENonEamId, ADOR.Recordset aRSBooksets, long byval alTransID, ADOR.Recordset optional byval aRSRuleVars)
Class: IQualifiedEventDefinition
Description:

Create a single qualified Event Line.

Inputs: byval alBusinessEventEntityId -

byval alProductId -
byval alEventModID -
alRuleId -
byval alJEID -
byval alJENonEarnId -
aRSBooksets -
byval alTransID -
optional byval aRSRuleVars -

Outputs: None
Returns: None

UpdateQualEventLine(long byval alQualEventId, long byval alBusinessEventId, long byval alProductId, long byval alEventModId, long byval alRulEID, long byval alJEID, long byval alJENonEarnId, string byval astrEntryName, ADOR.Recordset byval arsBooksets, long byref alTransID, ADOR.Recordset optional byval arsRuleVars)

Class: IQualifiedEventDefinition
Description:

Update a Qualified Event Line.

Inputs: byval alQualEventId -
byval alBusinessEventId -
byval alProductId -
byval alEventModId -
byval alRulEID -
byval alJEID -
byval alJENonEarnId -
byval astrEntryName -
byval arsBooksets -
byref alTransID -
optional byval arsRuleVars -

Outputs: None
Returns: None

DeleteQualEventLine(long byval alQualEventId, long byref alTransId)

Class: IQualifiedEventDefinition
Description:

Delete a pseicfic Qualified Event line.

Inputs: byval alQualEventId -
byref alTransId -

Outputs: None
Returns: None

long CreateRuleVar(long byval alVarTypeID, long byval alQualEventId, long byval alRuleLineId, long byval alVarSeqNum, long byval alPBEParmID, long byval alDBFieldID, string byval strConstantValue, long alORigRuleLine, long byref alTransId)

Class: IQualifiedEventDefinition

Description:
Create a Rule variable for a qualified Event line.
Inputs: byval alVarTypeID -
byval alQualEventId -
byval alRuleLineId -
byval alVarSeqNum -
byval alPBEParmID -
byval alDBFieldID -
byval strConstantValue -
alORigRuleLine -
byref alTransId -
Outputs: None
Returns: long

DeleteRuleVar(long byval alEntityId, long byref alTransId)
Class: IQualifiedEventDefinition
Description:
Delete a rule variable.
Inputs: byval alEntityId -
byref alTransId -
Outputs: None
Returns: None

IRuleDefinition Class

Description

This interface contains the methods required to create, update, and use a Rule in the Accounting Engine. This will maintain the Rule (Lookup table) entity: Rule name, description and the Rule lines that define the Rule.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

long CreateHdr(string byval astrRuleName, string byval astrRuleDesc, long byref alTransId)

Class:	IRuleDefinition
Description:	This will create a Rule in the Accounting Engine. Before an Event can refer to a Rule, it will be necessary to define (create) the Rule entity. Rule is a Rule header and lines. Return the Entity id for the Rule created, not for the Rule lines.
Inputs:	byval astrRuleName - byval astrRuleDesc - byref alTransId -
Outputs:	None
Returns:	long

Long AddRuleLine(Long byval alRuleId, Long byval alVerbLUID, String byval astrRuleDest, Long byval alLineSeq, ADOR.Recordset byval aRSRuleVars, Long byref alTransID)

Class:	IRuleDefinition
Description:	Add a single Rule line for a Rule. The Rule line is used to define the Rule Verb, Destination and the variables that need to be resolved to process the Rule.
Inputs:	byval alRuleId - byval alVerbLUID - byval astrRuleDest - byval alLineSeq -

byval aRSRuleVars -
byref alTransID -
Outputs: None
Returns: Long

DeleteRule(long byval alEntityId, long byref alTransId)
Class: IRuleDefinition
Description:
This will delete a Rule and all of its Rule lines in the Accounting Engine.
This is all or nothing behaviour. Rule lines can not be deleted if the Rule delete fails for any reason (including enforced referential integrity).

Inputs: byval alEntityId -
byref alTransId -
Outputs: None
Returns: None

String Ping()
Class: IRuleDefinition
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

UpdateHdr(long byval alEntityId, string byval astrName, string byval astrRuleDesc, long byref alTransNbr)
Class: IRuleDefinition
Description:
This will update one Rule Name or description in the Accounting Engine.

Inputs: byval alEntityId -
byval astrName -
byval astrRuleDesc -
byref alTransNbr -
Outputs: None
Returns: None

DeleteLine(long byval alEntityId, byref alTransId)
Class: IRuleDefinition
Description:

Delete one rule line.
Inputs: byval alEntityId -
byref alTransId -
Outputs: None
Returns: None

long CreateRuleVar(long byval alVarTypeID, long byval alRuleLineID, long byval alVarSeqNum, long byval alPBEParmID, Long byval alDBFieldID, string astrConstantValue, long byval alOrigRuleLine, long byref alTransID)
Class: IRuleDefinition
Description:
Create a single Rule Variable for a single Rule line.
Inputs: byval alVarTypeID -
byval alRuleLineID -
byval alVarSeqNum -
byval alPBEParmID -
byval alDBFieldID -
astrConstantValue -
byval alOrigRuleLine -
byref alTransID -
Outputs: None
Returns: long

DeleteVar(long byval alEntityID, long byref alTransID)
Class: IRuleDefinition
Description:
Delete a single variable that is no longer used by any Rule Lines.
Inputs: byval alEntityID -
byref alTransID -
Outputs: None
Returns: None.

PrivateAccess Methods

UpdateRuleLine(Long byval alEntityId, Long byval alVerbLUID, String byval astrRuleDest, Long byval alLineSeq, ADOR.Recordset byval aRSRuleVars, Long byref alTransId)
Class: IRuleDefinition
Description:
This is used to update one Rule Line. This can be used to change the Rule Line detail.
Inputs: byval alEntityId -
byval alVerbLUID -
byval astrRuleDest -
byval alLineSeq -
byval aRSRuleVars -
byref alTransId -
Outputs: None
Returns: None

IServiceClass

Description

This interface contains the methods required to "service" BSAEMaint. We need to review services against windows to make sure we can populate all fields that we have on existing windows.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

Long DoesSubledgerCodeExist(string byval astrSubledgerCodeExist)

Class:	IService
Description:	This will check for the existance of a subledger code and return the entityID for the subledger code if it is found.
Inputs:	byval astrSubledgerCodeExist -
Outputs:	None
Returns:	Long

ADOR.Recordset GetAllAccountingPeriods()

Class:	IService
Description:	Get all of the valid Accounting Periods and Dates that may be used in a Rule. e.g. CurrYear, PriorYear, Today, CurrMonth, etc.
Inputs:	None
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetAllBooksets()

Class:	IService
Description:	This will get all Booksets in the AE.
Inputs:	None
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetAllBusinessEvents()

Class: IService
Description: This will get all Business Events in the AE

Inputs: None
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetAllEventModifiers()
Class: IService
Description: This will get all Event Modifiers in the AE.
Inputs: None
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetAllJEs()
Class: IService
Description: This will get all Journal Entry headers in the AE. It does not return the debit / credit pairs.

Inputs: None
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetAllParms()
Class: IService
Description: This will get all Parameters defined in the AE.
Inputs: None
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetAllProductBusinessEvents()
Class: IService
Description:
This will get a list of all Product Business Events in the AE.
Inputs: None
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetAllProducts()
Class: IService
Description: This will get a list of all Products in the AE.
Inputs: None
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetAllRules()

Class:	IService
Description:	This will get a list of all Rules in the AE.
Inputs:	None
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetAllSubledgers()

Class:	IService
Description:	This will get a list of all Subledgers in the Chart of Accounts in the AE.
Inputs:	None
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetBooksetById(long byval alEntityId)

Class:	IService
Description:	This will get a Bookset in the AE.
Inputs:	byval alEntityId -
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetBusinessEventById(long byval alEntityId)

Class:	IService
Description:	This will get a Business Event in the AE.
Inputs:	byval alEntityId -
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetEventModifierById(long byval alEntityId)

Class:	IService
Description:	This will get an Event Modifier using the Entity Id of the Event Modifier.
Inputs:	byval alEntityId -
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetEventModifierLinesByID(Long byval alEntityId)

Class:	IService
Description:	This will return the Event modifier lines in an ADOR.Recordset.
Inputs:	byval alEntityId -
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetJEDetailsById(long byval alEntityId)

Class:	IService
Description:	This will get the JE header and the debit / credit pairs associated with a single JE in the AE.
Inputs:	byval alEntityId -
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetQEBooksetsByQELineID(long byval alEntityId)

Class:	IService
Description:	This will get the Booksets associated with a single Qualified Event line in the AE.
Inputs:	byval alEntityId -
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetQELinesByPBEt(long byval alProductEntityId, long byval alBusinessEventEntityId)

Class:	IService
Description:	This will get the Qualified Event Names and Line Sequence numbers associated with a single Event Product.
Inputs:	byval alProductEntityId - byval alBusinessEventEntityId -
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetRuleLinesByRuleId(long byval alEntityId)

Class:	IService
Description:	Get a single row from the Rule Field Lookup table using the field Alias.
Inputs:	byval alEntityId -
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetRuleById(long byval alEntityId)

Class:	IService
Description:	This will get a Rule, and all of its assocaited Rule lines in the AE.
Inputs:	byval alEntityId -
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetRuleVarsByRuleLineId(long byval alEntityId)

Class: IService
Description:
Get the Rule line variables associated with a single rule line.

Inputs: byval alEntityId -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetStreamTypeByID(long byval alEntityID)

Class: IService
Description:
Get the stream type by the stream type id.

Inputs: byval alEntityID -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetSubledgerById(long byval alEntityId)

Class: IService
Description: This will get a Subledger from the Chart of Accounts in the AE.
Inputs: byval alEntityId -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetSubledgerWithFilter(string byval astrColumn, string byval astrMatchPattern)

Class: IService
Description:
Return SI by SLcode using a SQL 'Like' Subledger code.
Inputs: byval astrColumn -
byval astrMatchPattern -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetSubIForGroup(long byval alEntityId)

Class: IService
Description:
This is return all of the subledgers in the Chart of Accounts for a single Subledger Group.

Inputs: byval alEntityId -

Outputs: None
Returns: ADOR.Recordset

String Ping()
Class: IService
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

ADOR.Recordset GetAllStreamTypes()
Class: IService
Description:
Get all Stream types for a dropdown of stream types.
Inputs: None
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetAllSicALER()
Class: IService
Description:
string containing the text: Asset, Liability, Expense, or Revenue.
Inputs: None
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetAllSicMemoGL()
Class: IS. rvice
Description:
String containing the value Memo or GL. This is used to populate the MEMOGL dropdown.

Inputs: None
Outputs: None
Returns: ADOh.Recordset

ADOR.Recordset GetPBEParmsByPBE(long byval alProductId, long byval alBusinessEventID)
Class: IService
Description:
PBE Product, Business Event.
Get the parms associated with a Product busienss Event.
Inputs: byval alProductId -

byval alBusinessEventID -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetQERuleVarsByQELineId(long byval alEntityId)
Class: IService
Description:

QE Qualified Event
Get all of the rule variables associated withg a qualified event line using the QE id.
Inputs: byval alEntityId -
Outputs: None
Returns: ADOR.Recordset

Boolean GetPBE(string byval astrProduct, string byval astrBusinessEvent, long byref alPBEID)
Class: IService
Description:

Get a single product business event by specifying the product and business event.

This is not completed yet.
Inputs: byval astrProduct -
byval astrBusinessEvent -
byref alPBEID -
Outputs: None
Returns: Boolean

IStreamDefinition Class

Description

This interface contains the methods required to create, update, and use streams in the Accounting Engine. This will maintain the Stream (Lookup table) entity: Stream name and description.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

long Create(string byval astrStreamName, String byval astrStreamDesc, long byref alTransId)

Class: IStreamDefinition
Description:

This will create a Stream in the Accounting Engine. Before any Asset Stream can be created it will be necessary to define (create) the Stream entity.

Inputs: byval astrStreamName -
byval astrStreamDesc -
byref alTransId -
Outputs: None
Returns: long

Delete(long byval alEntityId, long alTransID)

Class: IStreamDefinition
Description:

This will delete a Stream in the Accounting Engine. Before any Asset Stream can be deleted it will be necessary to verify this Stream is not currently being used by any Asset.

Inputs: byval alEntityId -
alTransID -
Outputs: None
Returns: None

Update(long byval alEntityId, String byval astrStreamDesc, long byval alTransID, String byval astrStreamName)

Class:	**IStreamDefinition**
Description:	**This will update the name or description for one Stream in the Accounting Engine.**
Inputs:	**byval alEntityId -** **byval astrStreamDesc -** **byval alTransID -** **byval astrStreamName -**
Outputs:	**None**
Returns:	**None**

String Ping()

Class:	**IStreamDefinition**
Description:	**Return a string indicating whether this object is instantiated.**
Inputs:	**None**
Outputs:	**None**
Returns:	**String**

ISubLedgerGroupDefinition Class

Description

This Interface is used to define subledger groups to the accounting engine.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

long AddSubledger(long byval alSLGroupId, long byval alSubledgerId, long byref alTransNbr)

Class: ISubLedgerGroupDefinition

Description:

Add a subledger to this Subledger Group.

- Error handler will trap invalid foreign keys.
- Error handler will trap duplicate entries in the SL_Group_Subledgers table.
- Insert into SL_Group_Subledgers values(ID)
- alTransNbr = Call LogTransId.
- Return ID as long

Inputs: byval alSLGroupId -
byval alSubledgerId -
byref alTransNbr -

Outputs: None

Returns: long

long Create(string byval astrName, string byval astrDesc, long byref alTransNbr)

Class: ISubLedgerGroupDefinition

Description:

This will create a new Subledger Group entity.

- Validate fields are not null.
- Insert into SI_Group values(astrName, astrDesc)
- alTransNbr = Call LogTransId.

Inputs: byval astrName -
byval astrDesc -
byref alTransNbr -
Outputs: None
Returns: long

Delete(long byval alEntityId, long byref alTransNbr)
Class: ISubLedgerGroupDefinition
Description:
Delete a Subledger Group from the Accounting Engine.

- Delete from SL_Group

- alTransNbr = Call LogTransId.

Inputs: byval alEntityId -
byref alTransNbr -
Outputs: None
Returns: None

RemoveSubledger(long byval alSubledgerGroupId, long byval alSubledgerId, long byref alTransNbr)
Class: ISubLedgerGroupDefinition
Description:
Remove a subledger from this Subledger Group.

- Delete from SL_Group_Subledgers where SQ_CHART_OF_ACCOUNT_ID = alSubledgerId and SQ_SL_GROUP_ID = alSLGroupId.
- alTransNbr = Call LogTransId.

Inputs: byval alSubledgerGroupId -
byval alSubledgerId -
byref alTransNbr -
Outputs: None
Returns: None

Update(Long byval alEntityId , String byval astrDesc, Long byref alTransNbr)
Class: ISubLedgerGroupDefinition
Description:
Update the description for a subledger group.
. update SL_GROUP
. alTransNbr = Call LogTransID

The name for a sl group can not be changed???
Inputs: byval alEntityId -
byval astrDesc -
byref alTransNbr -

Outputs: None
Returns: None

String Ping()

Class: ISubLedgerGroupDefinition
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

ISubledgerDefintion Class

Description

This interface contains the methods required to create, update, and use the Subledger Chart of Accounts in the Accounting Engine. This will maintain the Subledger (Lookup table) entity: Chart of Accounts, name, description.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

long Create(string byval astrRollupId, string byval astrTypeMemoGl, string abyval astrTypeALER, string byval astrSLCode, string byval astrActiveId, string byval astrSubLedgerName, long byval alTransferId, string byval astrCrossref, long byref alTransNbr)

Class:	**ISubledgerDefintion**
Description:	**This will create a Subledger in the Accounting Engine Chart of Accounts. Before any Asset can be refer to a Subledger, it will be necessary to define (create) the Subledger entity. This will return the entity id as a long.** **- Test for required fields (except for astrCrossRef).** **- Insert into SL_Chart_of_Accounts values** **- alTransNbr = Call LogTransId.** **- Return ID as long**
Inputs:	**byval astrRollupId -** **byval astrTypeMemoGl -** **abyval astrTypeALER -** **byval astrSLCode -** **byval astrActiveId -** **byval astrSubLedgerName -** **byval alTransferId -** **byval astrCrossref -** **byref alTransNbr -**
Outputs:	**None**
Returns:	**long**

Delete(long byval alEntityId, long byref alTransNbr)

Class: ISubledgerDefintion
Description:

This will delete a Subideger from the Chart of Accounts, in the Accounting Engine.

· Delete from SL_Chart_Accounts

· alTransNbr = Call LogTransId.

Inputs: byval alEntityId -
byref alTransNbr -
Outputs: None
Returns: None

Update(string byval astrSubLedgerName, string byval astrRollupId, string byval astrTypeMemoGl, string byval astrTypeALER, string byval strTransferId, string byval astrSLCode, string byval astrCrossRef, string byval astrActiveId, byval, long AlEntityId, long byref alTransNbr)

Class: ISubledgerDefintion
Description:

This will update one Subledger from the Chart of Accounts, in the Accounting Engine.

· Validate required field: all except astrCrossRef and alTransNbr.

· Update SL_Chart_of_Accounts

· alTransNbr = Call LogTransId.

Inputs: byval astrSubLedgerName -
byval astrRollupId -
byval astrTypeMemoGl -
byval astrTypeALER -
byval strTransferId -
byval astrSLCode -
byval astrCrossRef -
byval astrActiveId -
byval -
AlEntityId -
byref alTransNbr -
Outputs: None
Returns: None

String Ping()

Class: ISubledgerDefintion
Description: Return a string indicating whether this object is instantiated.
Inputs: None

Outputs: None
Returns: String

UpdateRS(ador.recordset byval arsSubledger)
Class: ISubledgerDefintion
Description:
Use ADOR recordset to add, uipdateo r delete a record from the database.
Inputs: byval arsSubledger -
Outputs: None
Returns: None

BSAudit Package Overview

Description

This package contains the business service classes required to support an enterprise model Audit trail.

This audit trail will contain a unique transaction number for each transaction, a facilityid which identifies the system that generated the transaction and entity information to identify the database entity that has changed.

Classes

IService
ITransaction

Subpackages

None

IServiceClass

Description

IService is used to retrievedata.

SQ_TRANSACTION_NBR is the database field used to identify the AE Transaction. There can be multiple rows in the Transaction table for each AE transaction.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

ADOR.Recordset GetTransByDate(date byval adteFrom, Date byval adteTo)

Class:	IService
Description:	This will get all transactions in the Audit Component for a given date range. This will return the transaction details as an ADOR.Recordset. Get Timestamp_Date = > adteFrom and =< adteTo Order by TIMESTAMP_DATE descending
Inputs:	byval adteFrom - byval adteTo -
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetTransByEntityId(long byval alEntityId)

Class:	IService
Description:	This will get a transaction entity in the Audit Component. This will return the details of a single row in the Transaction table as an ADOR.Recordset. alEntityId The Entity id of the specific row in the transaction table being returned.
Inputs:	byval alEntityId -
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetTransByTransNbr(Long byval alTransNbr)

Class:	IService

Description:	This will get a complete transaction in the Audit Component. This will return the details of a Transaction as an ADOR.Recordset. alTransNbr The transaction id of the specific transaction being returned.
Inputs:	byval alTransNbr -
Outputs:	None
Returns:	ADOR.Recordset

String Ping()

Class:	IService
Description:	Return a string indicating whether this object is instantiated.
Inputs:	None
Outputs:	None
Returns:	String

ITransaction Class

Description

TThis interface contains the methods required to create, and use the unique transaction id in the Audit Component. This component may be used by multiple components. This will be recorded on all Accounting Engine entities, when they are created or updated for a complete transaction audit trail. This will also be useful for enabling Undo functionality.

```
Public enum eAuditTransTypes
  ecAdd
  ecCreate
  ecDelete
  ecRemove
  ecUpdate
end enum
```

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

Long LogTrans(Long byval alFacilityId, eAuditTransTypes byval alTransType, string byval astrEntityName, long byval alEntityId, Long byref optional alTransNbr = 0)

Class:	ITransaction
Description:	This will generate a unique transaction number with details about the transaction for audit purposes and return the Entity id for the transaction generated in this function. The userid will be obtained from the MTS context Object. The date-timestamp will be obtained from the system. The alTransNbr will be generated if it is zero.

alTransType The type of database activity that was performed by this transaction as defined in the eAuditTransTypes enum.
astrEntityName The name of entity that is associated with the Entity ID stored for this transaction.
alEntityId The Entity ID for the entity involved in this activity.

- Get Userid from MTS context Object.
- Get Date-timestamp from system.
- If alTransNbr = zero
  tn = create new transaction number

Else
  tn = alTransNbr

· Insert into Audit_Transaction values(tn, userid, date-timestamp, astrEntityName, alEntityId, alTransType, alFacilityId)
· Return tn.

Inputs: byval alFacilityId -
byval alTransType -
byval astrEntityName -
byval alEntityId -
byref optional alTransNbr -
Outputs: None
Returns: Long

String Ping()
Class: ITransaction
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

BSCalendar Package Overview

Description

Classes

ICalendarDefinition
IService

Subpackages

None

ICalendarDefinition Class

Description

The Calendar will be used to identify the fiscal closing date for an entire Bookset and will be used to resolve key Activity dates used for periodic processes; e.g., Bank Holidays. If an Asset uses multiple Booksets then all of those Booksets are required to used the same Calendar.
The Calendar will be exposed to the Operational System.
Every Asset will be associated with one and only one Calendar.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

long CreateCalendar(String byval astrCalendarName, string byval astrCalendarDesc, integer byval aiFiscalYearStartMonth, integer byval aiFiscalYearStartDay, long byref alTransnbr)

Class:	ICalendarDefinition
Description:	This will create a Calendar in the Accounting Engine. Before any Asset can refer to a Calendar, it will be necessary to define (create) the Calendar entity.
Inputs:	byval astrCalendarName - byval astrCalendarDesc - byval aiFiscalYearStartMonth - byval aiFiscalYearStartDay - byref alTransnbr -
Outputs:	None
Returns:	long

CreateDates(long byval alCalendarEntityId, variant byval avFiscalStartMonths, byval byref alTranNbr)

Class:	ICalendarDefinition
Description:	This will create a series of dates for a single year in an existing Calendar in the calendar component. The fiscal month field on each date needs to be populated using the avFiscalStartMonths array passed into this method. A unique constraint on the Calendar Date and Calendar will ensure there are no duplicate dates for a calendar.
	alFiscalYear The fiscal year to be created

avFiscalStartMonths This is an array of the 11 dates which represent the fiscal start for each month, after the first month of the year. The first month of the year is derived from alFiscalYear and the Fiscal Start month and day on the FiscalCalendar.
adteFiscalYearEnd This is the last day of the fiscal year being created.
· Update_Date = Today
Update_Userid = objectcontext.OriginalCaller
· Insert Calendar_Date records for each day in alFiscalYear.
· Insert CALENDAR_ACTIVITY_DATE for each start date to create the junction between "FISCAL MONTH START" and the date
· return id

Inputs: byval alCalendarEntityId -
byval avFiscalStartMonths -
byval byref alTranNbr -
Outputs: None
Returns: None

Long CreateActivityType(string byval astrActivityTypeName, string astrActivityTypeDesc, Integer alReservedInd)
Class: ICalendarDefinition
Description: This will create a Calendar Activity Type in the Calendar component.

Type
· Update_Date = Today
Update_Userid = objectcontext.OriginalCaller
· Insert into Calendar_Activity_Type
· return id

Inputs: byval astrActivityTypeName -
astrActivityTypeDesc -
alReservedInd -
Outputs: None
Returns: Long

long AddDateActivity(long byval alActivityTypeEntityId, long byval alDateEntityId)
Class: ICalendarDefinition
Description: This is used to connect a Calendar Activity Type to a Calendar Date for a single Calendar.

The Entity Id for the Activity Type.
alDateEntityId The Entity Id for the date that is being associated with an Activity Type.
Update_Date = Today

Update_Userid = objectcontext.OriginalCaller
- Insert into Calendar_Date_Activty_Type values (alActivityTypeEntitytyId, alDateEntityId, Update_Date, Update_Userid)

Inputs: byval alActivityTypeEntityId -
byval alDateEntityId -
Outputs: None
Returns: long

DeleteDates(long byval alCalendarEntityId, integer alFiscalYear, long byref alTransNbr)

Class: ICalendarDefinition
Description: This will delete (for those mainframers among us "purge") Dates from a Calendar in the calendar component. This delete will cascade down to the CALENDAR_ACTIVITY_DATEs associated with it.

- Update_Date = Today
Update_Userid = objectcontext.OriginalCaller
- Delete all date records falling within fiscal year alFiscalYear.
- return id

Inputs: byval alCalendarEntityId -
alFiscalYear -
byref alTransNbr -
Outputs: None
Returns: None

DeleteActivityType(long byval alEntityId)

Class: ICalendarDefinition
Description: This will delete one Calendar Activity Type from the Calendar component. Referential integrity will not allow deletion if CALENDAR_ACTIVITY_DATE records exist for this Activity Type.

- Delete from Calendar_Activity_Type
Inputs: byval alEntityId -
Outputs: None
Returns: None

RemoveDateActivity(Long byval alDateEntityId, long byval alActivityEntityId, long byval alTransnbr)

Class: ICalendarDefinition
Description: This will disassociate a Calendar_Activity_Type from a Calendar_Date by deleting the corresponding

Calendar_Activity_Date record from the Calendar component.

Inputs: byval alDateEntityId -
byval alActivityEntityId -
byval alTransnbr -
Outputs: None
Returns: None

UpdateActivityType(long byval alEntityId, string byval astrName, string byval astrDesc, integer byval alReservedInd)

Class: ICalendarDefinition
Description: This will update a Calendar ActivityType in the Calendar component.

Inputs: byval alEntityId -
byval astrName -
byval astrDesc -
byval alReservedInd -
Outputs: None
Returns: None

String Ping()

Class: ICalendarDefinition
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

UpdateCalendar(String byval astrCalendarName, string byval astrCalendarDesc, long byref alTransnbr, integer byval alReservedInd)

Class: ICalendarDefinition
Description:

This will update a Calendar in the Accounting Engine. Before any Asset or Corp can refer to a Calendar, it will be necessary to define (create) the Calendar entity.

Inputs: byval astrCalendarName -
byval astrCalendarDesc -
byref alTransnbr -
byval alReservedInd -
Outputs: None
Returns: None

ServiceClass

Description

This provides the services for the Calendar component

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

ADOR.Recordset GetAllCalendars()

Class:	IService
Description:	Return Calendar name and description for all Calendars.
Inputs:	None
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetAllActivityTypes()

Class:	IService
Description:	Return activity name and description for all Activity Types. This will be used for drop down list boxes
Inputs:	None
Outputs:	None
Returns:	ADOR.Recordset

Integer GetFiscalMonth(long byvalAlEntityId, date byval adteDate)

Class:	IService
Description:	Return the fiscal month for a specific date.
Inputs:	byvalAlEntityId - byval adteDate -
Outputs:	None
Returns:	Integer

ADOR.Recordset GetAllActivitiesByDate(long byval alEntityId, date astrFromDate, date asteToDate)

Class:	IService
Description:	Return Calendar, Activity Type and Activity Dates for a date range. This will be used to populate the calendar interface.

Inputs:	byval alEntityId - astrFromDate - asteToDate -
Outputs:	None
Returns:	ADOR.Recordset

String Ping()

Class:	IService
Description:	Return a string indicating whether this object is instantiated.
Inputs:	None
Outputs:	None
Returns:	String

ADOR.Recordset GetAllActivityByDateRange(long byval alEntityId, date astrFromDate, date asteToDate, long byval alActivityTypeId)

Class:	IService
Description:	Return Calendar,and Activity Dates for one Activity Type and a date range.
Inputs:	byval alEntityId - astrFromDate - asteToDate - byval alActivityTypeId -
Outputs:	None
Returns:	ADOR.Recordset

BSCurrency Package Overview

Description

Classes

ICurrencyDefinition
IService

Subpackages

None

ICurrencyDefinition Class

Description

This interface contains the methods required to create, update, and use a Currency in the Accounting Engine. This will maintain the Currency, Rounding Rules and Currency Rate (Lookup table) entities: Currency name, description, rates, rounding rules,description. It will be necessary to define a valid Rounding Rule before creating a Currency.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

long CreateCurrency(String byval astrCurrencyName, string byval alRoundingRuleEntityId, long byval alTransNbr)

Class:	ICurrencyDefinition
Description:	This will create a Currency in the Accounting Engine. Before any Asset can be refer to a Currency, it will be necessary to define (create) the currency and a rounding rule.
Inputs:	byval astrCurrencyName - byval alRoundingRuleEntityId - byval alTransNbr -
Outputs:	None
Returns:	long

CreateRate(Long byval alFromEntityId, long byval alToEntityId, date byval adteEffectiveDate, string byval asConversionRate : single byval astrConversionSrc, long byref alTransNbr)

Class:	ICurrencyDefinition
Description:	This will create a Currency Rate in the Accounting Engine..Let's discuss. Do we have to get the From and To Currency Id before we create a rate.
Inputs:	byval alFromEntityId - byval alToEntityId - byval adteEffectiveDate - byval astrConversionSrc - byref alTransNbr -
Outputs:	None
Returns:	None

long CreateRoundingRule(String byval astrName, string byval astrDescription, long byval alRoundingTypeId, integer byval aintRoundingPos)

Class:	ICurrencyDefinition
Description:	This will create a Currency in the Accounting Engine. Before any Asset can be refer to a Currency, it will be necessary to define (create) the currency and a rounding rule.
Inputs:	byval astrName - byval astrDescription - byval alRoundingTypeId - byval aintRoundingPos -
Outputs:	None
Returns:	long

UpdateCurrency(Long byval alEntityId, VariantArray byval avCurrencyData, long alTransNbr)

Class:	ICurrencyDefinition
Description:	This will update one Currency Exchange Rate in the Accounting Engine.
Inputs:	byval alEntityId - byval avCurrencyData - alTransNbr -
Outputs:	None
Returns:	None

UpdateRate(Long byval alEntityId, string astrConversionSource, long byval alConversionRate, long byref alTransNbr)

Class:	ICurrencyDefinition
Description:	This will update one Currency Exchange Rate in the Accounting Engine.
Inputs:	byval alEntityId - astrConversionSource - byval alConversionRate - byref alTransNbr -
Outputs:	None
Returns:	None

UpdateRoundingRule(Long byval alEntityId, long byval alIndRoundTo, string byval astrRuleDescription, integer byval aintRoundToDecimal, long byref alTransNbr)

Class:	ICurrencyDefinition
Description:	This will update one Rounding Rule in the Accounting Engine.
Inputs:	byval alEntityId - byval alIndRoundTo - byval astrRuleDescription - byval aintRoundToDecimal - byref alTransNbr -
Outputs:	None

Returns: None

String Ping()

Class: ICurrencyDefinition
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

DeleteCurrency(long byval alEntityId, long byref alTransNbr)

Class: ICurrencyDefinition
Description: This will delete a Rounding Rule in the Accounting Engine. Referential integrity needs to be enforced.
Inputs: byval alEntityId -
byref alTransNbr -
Outputs: None
Returns: None

DeleteRoundingRule(long alEntityId, long byref alTransNbr)

Class: ICurrencyDefinition
Description: This will delete a Rate Conversion from the Accounting Engine.
Inputs: alEntityId -
byref alTransNbr -
Outputs: None
Returns: None

DeleteRate(long byval alEntityId, long byref alTransNbr)

Class: ICurrencyDefinition
Description: This will delete a Rate Conversion from the Accounting Engine.
Inputs: byval alEntityId -
byref alTransNbr -
Outputs: None
Returns: None

IServiceClass

Description

This provides the services for the Currency component

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

ADOR.Recordset GetAllCurrencies()

Class:	IService
Description:	This will get Currency details for all currencies defined to the AE. This will include name and rounding rule.
Inputs:	None
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetAllRoundingRules()

Class:	IService
Description:	This will get all of the rounding rules defined to the AE.
Inputs:	None
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetRoundingRule(long byval alentityid)

Class:	IService
Description:	This will get a Rounding Rule in the Accounting Engine.
Inputs:	byval alentityid -
Outputs:	None
Returns:	ADOR.Recordset

String Ping()

Class:	IService
Description:	Return a string indicating whether this object is instantiated.
Inputs:	None
Outputs:	None
Returns:	String

Long GetRoundingRuleId(alEntityId)

Class:	**IService**
Description:	**This will return the entity id for a Rounding Rule in the Accounting Engine.**
Inputs:	**alEntityId -**
Outputs:	**None**
Returns:	**Long**

GetRate(long byval alentityId)

Class:	**IService**
Description:	**This will get a Currency Rate in the Accounting Engine.**
Inputs:	**byval alentityId -**
Outputs:	**None**
Returns:	**None**

BSFinancialOrg Package Overview

Description

This interface contains the methods required to define a Business to the Accounting Engine.

Classes

IBusinessDefinition
ICorporationDefinition
IOfficeDefinition
IService

Subpackages

None

IBusinessDefinition Class

Description

This interface contains the methods required to define a Business to the Accounting Engine.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

long Create(string byval astrBusinessName, string byval astrBusinessDescription, long byref alTransId)

Class:	IBusinessDefinition
Description:	This will create a business in the Accounting Engine. Before any Office can be defined within a Business, it will be necessary to define (create) the Business entity. Insert the Business
Inputs:	byval astrBusinessName - byval astrBusinessDescription - byref alTransId -
Outputs:	None
Returns:	long

Delete(long byval alEntityId, long byref alTransId)

Class:	IBusinessDefinition
Description:	This will delete a Business from the Accounting Engine. Referential integrity needs to be enforced between this and the Corporation and the office. Delete the business
Inputs:	byval alEntityId - byref alTransId -
Outputs:	None
Returns:	None

Update(long byval alEntityId, string astrBusinessName, byval astrBusinessDescription, long byref alTransId)

Class:	IBusinessDefinition

Description: This will update a Business defined to the Accounting Engine.

- lstrUserId = context.security.getoriginalcaller
lstrDate = Date
- Update Business_AE

Inputs: byval alEntityId -
astrBusinessName -
byval astrBusinessDescription -
byref alTransId -

Outputs: None

Returns: None

String Ping()

Class: IBusinessDefinition

Description: Return a string indicating whether this object is instantiated.

Inputs: None

Outputs: None

Returns: String

ICorporationDefinition Class

Description

This interface contains the methods required to define a Corporation to the Accounting Engine. This will maintain the Corporation (Lookup table) entity.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

Create(string byval astrCorpName, string byval astrCorpDescription, long byval alCalendarId, long byref alTransId)

Class:	ICorporationDefinition
Description:	This will create a Corporation in the Accounting Engine. Before any Asset, Office, or Business can refer to a Corporation., it will be necessary to define (create) the Corporation entity. This will return the Entity Id as a long. alCalendarId The id of a fiscal calendar to be associated with this corporation. · lstrUserId = context.security.getoriginalcaller lstrDate = Date · Insert into Corp_Org
Inputs:	byval astrCorpName - byval astrCorpDescription - byval alCalendarId - byref alTransId -
Outputs:	None
Returns:	None

Delete(Long byval alEntityId, long byref alTransId)

Class:	ICorporationDefinition
Description:	This will Delete a Corporation defined to the Accounting Engine. Referential integrity needs to be enforced. · delete from Corp_Org
Inputs:	byval alEntityId - byref alTransId -
Outputs:	None

Returns: None

Update(Long byval alEntityId, string byval astrCorpName, string byval astrCorpDescription, long alCalendarID, long byref alTransId)

Class: ICorporationDefinition
Description: This will update a Corporation defined to the Accounting Engine.

- lstrUserid = context.security.getoriginalcaller
lstrDate = Date
- Update Corp_Org

Inputs: byval alEntityId -
byval astrCorpName -
byval astrCorpDescription -
alCalendarID -
byref alTransId -
Outputs: None
Returns: None

String Ping()

Class: ICorporationDefinition
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

IOfficeDefinition Class

Description

This interface contains the methods required to define an Office to the Accounting Engine. This will maintain the Office (Lookup table) entity.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

AddCorp(long byval alOfficeId, byval alCorpId, long byref alTransId)

Class: IOfficeDefinition
Description:
This will add a junction relationship between an Office and a Corp.

- lstrUserId = context.security.getoriginalcaller
lstrDate = Date
- Insert into Office_Corp

Inputs: byval alOfficeId -
byval alCorpId -
byref alTransId -
Outputs: None
Returns: None

long Create(string byval astrOfficeName, string byval astrOfficeDescription, long byval alBusinessId, variant byval AVCorpIds)

Class: IOfficeDefinition
Description:
This will create an Office in the Accounting Engine. Before any Asset can refer to an Office, it will be necessary to define (create) the Office entity. This will return the Entity id as a long.

- lstrUserId = context.security.getoriginalcaller
lstrDate = Date
- Insert into Office
- Insert rows into Corp_Office for each corporation in the avCorpIds array.
- return id

Inputs: byval astrOfficeName -
byval astrOfficeDescription -

byval alBusinessId -
byval AVCorpIds -
Outputs: None
Returns: long

Delete(long byval AlEntityId, long byref alTransID)
Class: IOfficeDefinition
Description:

This will Delete an Office defined to the Accounting Engine. Referential integrity needs to be enforced between this and the Corporation and the Asset.

· alTransId = Call LogTrans to log the transaction and get the associated transaction number.
· delete from Office

Inputs: byval AlEntityId -
byref alTransID -
Outputs: None
Returns: None

RemoveCorp(long byal alOfficeId, byval alCorpId, long byref alTransID)
Class: IOfficeDefinition
Description:

This will remove a junction relationship between an Office and a Corp.

· Delete from Office_Corp

Inputs: byal alOfficeId -
byval alCorpId -
byref alTransID -
Outputs: None
Returns: None

Update(long byval alEntityId, string byval astrOfficeName, string byval astrOfficeDescription, long byval alBusinessId, string byval astrBusinessDescription)
Class: IOfficeDefinition
Description:

This will update one Office in the Accounting Engine.

· alTransId = Call LogTrans to log the transaction and get the associated transaction number.
· lstrUserId = context.security.getoriginalcaller
lstrDate = Date
· Update Office

- Insert or delete rows from Corp_Office for each corporation in the avCorpIds array.

Inputs: byval alEntityId -
byval astrOfficeName -
byval astrOfficeDescription -
byval alBusinessId -
byval astrBusinessDescription -
Outputs: None
Returns: None

String Ping()
Class: IOfficeDefinition
Description:
Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

ProtectedAccess Methods

AddCorpInt(ascFinOrgDataClas byref aDataClass, MTxAS.ObjectContext ByRef aContext, Long ByRef alOfficeId, Long ByRef alCorpId, Long ByRef alTransNbr)
Class: IOfficeDefinition
Description:
Associate a corp with an office. This PRIVATE sub is called from other subs in the interface.
Inputs: byref aDataClass -
ByRef aContext -
ByRef alOfficeId -
ByRef alCorpId -
ByRef alTransNbr -
Outputs: None
Returns: None

IService Class

Description

This provides the services for the Financial Organization component.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

ADOR.Recordset GetAllAllBusinesses()

Class:	IService
Description:	This will retrieve all businesses from the accounting engine returning them in a recordset.
Inputs:	None
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetAllCorporations()

Class:	IService
Description:	This will retrieve all corporations from the accounting engine returning them in a recordset.
Inputs:	None
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetAllAllOffices()

Class:	IService
Description:	This will retrieve all offices from the accounting engine returning them in a recordset.
Inputs:	None
Outputs:	None
Returns:	ADOR.Recordset

ADOR.Recordset GetBusinessByID(long byval alEntityId)

Class:	IService

Description: This will get a Business defined to the Accounting Engine using the business's entity id.

Inputs: byval alEntityId -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetCorporationById(long byval alEntityId)
Class: IService
Description: This will get a Corporation defined to the Accounting Engine using the corporation's entity id.

Inputs: byval alEntityId -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetOfficeById(long alEntityId)
Class: IService
Description: This will get an Office defined to the Accounting Engine using the office's entity id.

Inputs: alEntityId -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetCorpsForOffice(Long byval alOfficeId)
Class: IService
Description: This will get all Corps associated with an Office.
Inputs: byval alOfficeId -
Outputs: None
Returns: ADOR.Recordset

ADOR.Recordset GetOfficesForCorp(long byval alCorpId)
Class: IService
Description: This will get all Corporations associated with an Office.
Inputs: byval alCorpId -
Outputs: None
Returns: ADOR.Recordset

String Ping()
Class: IService
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

BSImportExport Package Overview

Description

Classes

IImportExport

Subpackages

None

IImportExport Class

Description

This interface is used for input and output operations which must be performed in large numbers. These import methods will accept a file as input. Export will create a file as output. This interface will be useful for creating reporting files and for conversion activities. The Long returned from each of these method calls will contain the number of records passed into or out of the engine.
This interface will be used for exporting date for the General Ledger.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

Long ExportAssetDetails(String byval astrFileName, string byval astrCorpName)

Class:	**IImportExport**
Description:	**Export Asset data for all assets to an external file. This will return the number of assets written to astrFilename, which will also be included in the file. This will return all assets in a corporation or 'ALL' assets.**
Inputs:	**byval astrFileName - byval astrCorpName -**
Outputs:	**None**
Returns:	**Long**

Long ExportAssetGroupAssetDetails(long byval alFacilityId, string byval astrCorpName, String byval astrExtAssetGroupType, String byval astrExtAssetGroupRef, String byval astrFileName)

Class:	**IImportExport**
Description:	**Export Asset data for all assets in designated group, or all asset, to an external file. astrFilename will contain the number of assets written to it and the asset group name followed by the data. This will return all assets in a corporation or 'ALL' assets.**
Inputs:	**byval alFacilityId - byval astrCorpName - byval astrExtAssetGroupType - byval astrExtAssetGroupRef - byval astrFileName -**

Outputs: None
Returns: Long

Long ExportSLBalance(Long byval alBookSetId, String byval astrFileName, Date byval adtePeriod)

Class: IImportExport
Description: Export Sub ledger balances for an entire bookset, for a single period, to an external file. This will return the number of subledger balances written to astrFilename.

alBookSetId The bookset which will be exported.
adtePeriod The fiscal period used to filter this export.
astrFilename The output filename.

Inputs: byval alBookSetId -
byval astrFileName -
byval adtePeriod -
Outputs: None
Returns: Long

Long ExportSLDetailForDateRange(Long byval alBooksetID, Date byval adteFrom, Date byval adteTo, String byval astrFileName)

Class: IImportExport
Description: Export Sub ledger detail, by asset group, for an entire bookset, for a date range, to an external file. This will return the number of subledger details written to astrFilename.

Inputs: byval alBooksetID -
byval adteFrom -
byval adteTo -
byval astrFileName -
Outputs: None
Returns: Long

Long ExportSLDetailForPeriod(Long byval alBooksetId, Date byval adtePeriod, String byval astrFileName)

Class: IImportExport
Description: Export Sub ledger detail, by Asset Group, for an entire bookset, for a period, to an external file. This will return the number of subledger details written to astrFilename.

Inputs: byval alBooksetId -
byval adtePeriod -
byval astrFileName -
Outputs: None

Returns: Long

Long ExportStreamByGeneration(Integer byval alStreamGen, Long byval alBookSetID, String byval astrFileName)

Class: IImportExport
Description: Export Streams of one generation(current, original,etc.), by Asset, for an entire bookset or 'All' Booksets, to an external file. This will return the number of Streams written to astrFileName

Inputs: byval alStreamGen -
byval alBookSetID -
byval astrFileName -
Outputs: None
Returns: Long

Long ExportStreamByName(String byval astrStreamName, Integer byval alStreamGen, Long byval alBookSetId, String byval astrFileName)

Class: IImportExport
Description: Export Streams of one name (Rent, Income,etc), by Asset, for an entire bookset or 'All' Booksets, to an external file. This will return the number of Streams written to astrFileName
Inputs: byval astrStreamName -
byval alStreamGen -
byval alBookSetId -
byval astrFileName -
Outputs: None
Returns: Long

Long ImportAssetDetails(String byval astrFileName)

Class: IImportExport
Description: Import Asset data for multiple assets from an external file. This will include the number of assets to be read from astrFilename.
Inputs: byval astrFileName -
Outputs: None
Returns: Long

Long ImportAssetGroupAssetDetails(String byval astrFileName)

Class: IImportExport
Description: Import one Asset group from astrFilename. This file will contain asset group name, AE asset id and the number of assets to be read. If the asset group exists then asset will appended to existing asset group. If the

asset group does not exist, the asset group will be created and the assets will be added to it.

Inputs: byval astrFileName -
Outputs: None
Returns: Long

Long ImportAssetGroupUDFs(Long byval astrFileName)
Class: IImportExport
Description: Import Asset Group UDF's from astrFilename. This file will contain asset group name and UDF name / value pairs.
Inputs: byval astrFileName -
Outputs: None
Returns: Long

long ImportAssetUDFs(string byval astrFileName)
Class: IImportExport
Description: Import Asset UDF from astrFilename. This file will contain external asset reference and asset UDF name / value pairs.
Inputs: byval astrFileName -
Outputs: None
Returns: long

Long ImportCurrencyRates(Long byval astrFileName)
Class: IImportExport
Description: Import currency conversion rates from astrFilename. This file will contain currency type, country, rates, and effective dates

Inputs: byval astrFileName -
Outputs: None
Returns: Long

Long ImportCurrencyRoundingRules(Long byval astrFileName)
Class: IImportExport
Description: Import currency conversion rounding rules from astrFilename. This file will contain currency type, country, and rounding rules.

Inputs: byval astrFileName -
Outputs: None
Returns: Long

Long ImportStreamByGeneration(Integer byval aiStreamGen, Long byval aiBookSetId, Stream byval astrFileName)

Class:	**IImportExport**
Description:	**Import Streams of one generation(current, original,etc.), by Asset, for an entire bookset or 'All' Booksets, from an external file. This will return the number of Streams created**
Inputs:	**byval aiStreamGen - byval aiBookSetId - byval astrFileName -**
Outputs:	**None**
Returns:	**Long**

Long ImportStreamByName(String byval astrStreamName, Integer byval aiStreamGen, Long byval aiBookSetId, String byval astrFileName)

Class:	**IImportExport**
Description:	**Import Streams of one name (Rent, Income,etc), by Asset, for an entire bookset or 'All' Booksets, from an external file. This will return the number of Streams created.**
Inputs:	**byval astrStreamName - byval aiStreamGen - byval aiBookSetId - byval astrFileName -**
Outputs:	**None**
Returns:	**Long**

BSUDF Package Overview

Description

Classes

IService
IUDF

Subpackages

None

IService Class

Description

This provides the services for the BSUDF component

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

ADOR.Recordset GetAllUDFNames()

Class: IService
Description:

This will get the list of all of the UDFs defined to the Accounting Engine.

Inputs: None
Outputs: None
Returns: ADOR.Recordset

String GetUDFValue(string byval astrName, long alEntityId, long alInstanceId)

Class: IService
Description:

This will get the value of a User Defined Field associated with an entity.

astrName The User defined field to be returned.
alEntityId The id of the entity for which the UDF is being returned. (e.g. id for "asset")
alInstanceId The id of the specific entity instance for which the UDF is being returned. (e.g. id for asset #123)

Inputs: byval astrName -
alEntityId -
alInstanceId -
Outputs: None
Returns: String

ADOR.RecordSet GetAllUDFTableNames()

Class: IService
Description:

List of all Table Names available for use by UDF.
Inputs: None
Outputs: None
Returns: ADOR.RecordSet

String Ping()
Class: IService
Description:
Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

IUDF Class

Description

This interface contains the methods required to define UDF's (User Defined Fields) to the Accounting Engine. This will allow operational systems to define their own variables to be associated with specific instances of entity with the UDF Component. Each UDF will consist of a name/value pair.
Note, the operational systems will be responsible for passing the data values required for a UDF.
Accounting / Finance will define the UDF Names to help prevent the proliferation of UDF's, but a system Actor will actually populate the UDF Names table.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

CreateName(string byval astrName, String byval astrDesc, String byval astrTableName)

Class: IUDF
Description:

This will create a User Defined Field Name and Description. The UDF Name will be associated with the UDF Name - Value pairs held for specific entities.

astrName The name of the UDF to be created.
astrDescription.
astrTableName The name of the database table that will be used with the EntityId that is passed in to maintain a Name / Value pair.

Inputs: byval astrName -
byval astrDesc -
byval astrTableName -
Outputs: None
Returns: None

DeleteName(string byval astrName)

Class: IUDF
Description:

This will delete a User Defined Field Name. The UDF Name can not be deleted until all of the Name / Value pairs referring to this UDF have been successfully removed.

Inputs: byval astrName -
Outputs: None
Returns: None

DeleteValue(string byval astrName, long byval alInstanceId)
Class: IUDF
Description:
This will remove a user defined field value.

astrName The name of the UDF for which a value is being deleted.

alInstanceId The id of the specific entity instance for which the UDF is being deleted. (e.g. id for asset #123)

Inputs: byval astrName -
byval alInstanceId -
Outputs: None
Returns: None

UpdateValue(string byval astrName, long byval alInstanceId, string byval astrValue)
Class: IUDF
Description:
This will update the value of a User Defined Field associated with an entity.

astrName The name of the UDF for which a value is being updated.
alInstanceId The id of the specific entity instance for which the UDF is being deleted. (e.g. id for asset #123)
astrValue The value to be updated for this UDF

Inputs: byval astrName -
byval alInstanceId -
byval astrValue -
Outputs: None
Returns: None

String Ping()
Class: IUDF
Description: Return a string indicating whether this object is instantiated.
Inputs: None
Outputs: None
Returns: String

CreateValue(string byval astrName, long byval alInstanceId, string byval astrValue)

Class: IUDF

Description:

This will add a User_Defined_Field_Value for a specific user defined field name, associated with a specific entity.

astrName The name of the UDF for which a value is being added.
alInstanceId The id of the specific entity instance for which the UDF is being defined. (e.g. the id for asset # 123)
astrValue The value to be added to the UDF.

Inputs: byval astrName -
byval alInstanceId -
byval astrValue -

Outputs: None

Returns: None

BSEventProc Package Overview

Description

Classes

IEventProc
IService
IPostSL

Subpackages

None

IEventProc Class

Description

The IEvent processing interface contains methods used in calls to the Accounting Engine which require the creation of Journal Entries. Information passed to this interface will be combined with data stored in the Accounting Engine to process through Events defined inside the Accounting Engine.
This interface is fundamental to transaction processing between the Operational system and the Accounting Engine.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

long DoAssetEvent(Long byval alEventID, long byval alProductId, long byval AssetId, long byref alTransID, ParamArray opt byval ParmArray ())

Class:	IEventProc
Description:	Process an Event using an Asset. Parm Array consists of: Standard Input Parms(variant array) of which there must be at least one (an Asset or Asset Group ID). Event Modifiers (as many as are applicable). Format as Modifier ID and any inpt parms. Each event modifier is a variant array.
Inputs:	byval alEventID - byval alProductId - byval AssetId - byref alTransID - opt byval ParmArray () -
Outputs:	None
Returns:	long

String Ping()

Class:	IEventProc
Description:	Return a string indicating whether this object is instantiated.
Inputs:	None
Outputs:	None
Returns:	String

long VerifyParameters(Variant byval InputParmArray)

Class: IEventProc

Description:

Get all of the rows in the product_business_event_parm joined with the parm.
Check that the same number of Parms were passed in the Array of parameters as there are rows in the table. Only count parms that are event parms nad not event parms.
Event parms will be checked later.
Verify that all parms are not empty (null or space/zero is ok)

Inputs: byval InputParmArray -

Outputs: None

Returns: long

---

long VerifyEventModifier(Variant byval EventModifierVariantArray)

Class: IEventProc

Description:

Get the row in the Event_Modifier_table corresponsding to the passed ID of the Event Mod variant Array joined with the Event_Modifier_Line table.
For each Event MOdifier Line table row with SQ_PARM_ID not null, verify that a value was passed in the Event MOdifier Variant Array (a non empty variant value)

Inputs: byval EventModifierVariantArray -

Outputs: None

Returns: long

long SetUpEventModifierTT()

Class: IEventProc

Description:

Validate the Event Modifier.
If the value to be checked is a database field, execute SQL to obtain the value of that database field.
If it's a Parm, we already have the value on the Parm Value collection. Check the value obtained against the condition set. If it is true, check the next event modifier line until a FALSE condition is encountered. If you process the last Event Modifier line without hitting a FALSE condition, set the value of the Event Modifier Truth Table collection(indexed by SQ_EVENT_MODIFIER_ID) to TRUE, otherwise if any FALSE is found, set it to FALSE.

Inputs: None

Outputs: None

Returns: long

SetUpParmCollection()

Class:	**IEventProc**
Description:	**None**
Inputs:	**None**
Outputs:	**None**
Returns:	**None**

IService Class

Description

The service classs for the event processor.

This is where all of the retrieval only methods are enscapsulated for the event processor business service.

Many of these services will actually call to other IService routines in other DLL's. Rather than have the individual interfaces in the BSAEEventProc call the individual services, this IService interface will act like a controller class with the intelligence to decide how to get the data. It can either obtain itself or by calling it from a brother routine, as appropriate.

This will aid maintainability and maximize re-use.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

VerifyEventProduct()

Class:	IService
Description:	None
Inputs:	None
Outputs:	None
Returns:	None

VerifyAsset()

Class:	IService
Description:	None
Inputs:	None
Outputs:	None
Returns:	None

VerifyAssetProduct()

Class:	IService
Description:	None
Inputs:	None
Outputs:	None
Returns:	None

ObtainRuleInformation()

Class: IService
Description: None
Inputs: None
Outputs: None
Returns: None

ObtainQELineData()
Class: IService
Description: None
Inputs: None
Outputs: None
Returns: None

VerifyBooksets()
Class: IService
Description: None
Inputs: None
Outputs: None
Returns: None

ObtainRuleVars()
Class: IService
Description: None
Inputs: None
Outputs: None
Returns: None

GetDBFieldValue()
Class: IService
Description: None
Inputs: None
Outputs: None
Returns: None

GetJEDetails()
Class: IService
Description: None
Inputs: None
Outputs: None
Returns: None

VerifyBooksets()
Class: IService
Description: None
Inputs: None

Outputs:	None
Returns:	None

CheckExistingSubledgerBalance()

Class:	IService
Description:	None
Inputs:	None
Outputs:	None
Returns:	None

GetCorp()

Class:	IService
Description:	None
Inputs:	None
Outputs:	None
Returns:	None

IPostSL Class

Description

This is a controller class used to post the Subledgers.

PublicAccess Attributes

ProtectedAccess Attributes

PrivateAccess Attributes

PublicAccess Methods

Initialize()

Class:	IPostSL
Description:	
Inputs:	None
Outputs:	None
Returns:	None

Terminate()

Class:	IPostSL
Description:	None
Inputs:	None
Outputs:	None
Returns:	None

long PostAmount(long byval alAssetId, long alQualEventLineID, currency acurTXNAmount, Enum? aEnumAction)

Class:	IPostSL
Description:	None
Inputs:	byval alAssetId - alQualEventLineID - acurTXNAmount - aEnumAction -
Outputs:	None
Returns:	long

CreateSLDetail(long byval SLBalanceID, long byval alJEID, long alProductID, long alBankID, currency acurTXNAmount, string astrDRCRIND, long alCOAID, string astrPostPeriod)

Class:	IPostSL
Description:	None
Inputs:	byval SLBalanceID -

APPENDIX B
ACCOUNTING ENGINE USE CASE DEFINITIONS

Use Case: AE Maintenance

Scenario: Bookset Definition

Every set of sub-ledger entries is organized around a set of books. Each set of books or trial balance) can be identified using Book Set ID or Name. The Book Set can be used to segregate trial balance information like: tax and book entries, dual accounting or specific portfolios for portfolio acquisitions.

Bookset definition consists of creating, viewing, updating, and deleting Booksets in the AE.

Role:

Add, update and delete need to be an easily accessible, secured functions available to the AE administrator.

Viewing needs to be available to all AE users.

Frequency / Volume:

- There will only be a handful of booksets that will rarely be changed.

Input / Output Data:

Fiscal Calendar will need to be defined before the bookset is created to correctly identify the fiscal close calendar.

Tax type indicator values: Tax, Book, All

Report type indicator: Local, U.S. or Both

Book-set name should be unique.

Description.

Status and Status date. The Business Service maintains both these fields.

Business Rules:

Only one bookset can have a given bookset name.

All fields are required except for bookset description.

Create New Bookset

It should be possible to create new booksets for use in the Accounting Engine at any time

- The current date should be used for the Status Date of a new bookset.

Viewing a Bookset

It should be possible to view any bookset that exists in the accounting engine including all fields associated with the bookset.

Update a Bookset

It should be possible to update a bookset in the accounting engine. A bookset can not be updated until after it has been viewed.

All of the fields that are on the bookset should be available for update except for the bookset status that will be maintained automatically by the system. The bookset status should always reflect the date the last status changed was processed for this bookset.

Delete a Bookset

It should be possible to delete a bookset. A Bookset can only be deleted if there are no existing Assets associated with the Booksets. A decision needs to be made on how to enforce referential integrity for qualified events that use this bookset.

A bookset can not be deleted until the Bookset details have been viewed.

Issues

Notes

Use Case: AE Maintenance

Scenario: Product Definition

The specific accounting entries required for each transaction on a deal will be driven by the deal Product type. Product type is used to define specific accounting support that is required by FASB, the IRS and the SBA.

Product Maintenance consists of creating, viewing, updating and deleting Products in the Accounting Engine.

ATLAS will need the ability to query the Accounting Engine for a list of all existing products.

Role:

Add, update and delete need to be an easily accessible, secured functions available to the AE administrator.

Viewing needs to be available to all AE users.

Frequency / Volume:

- There will only be a handful of booksets that will rarely be changed.

Input / Output Data:

Product Name is unique and required.

Description is optional.

Last update date is system generated.

Last update user-id is system generated.

Business Rules:

Product Name needs to be unique.

Create New Product

It should be possible to enter the details for this Product: Product name and product description and create a new product at any time.

Viewing a Product

1. Find a Product and show all Product details. Finding the Product is a precursor to deleting or updating the Product. The user will normally know the Product Name for a Product they wish to view.

Update a Product

Update Product details: Product Name and description after the Product details have been viewed.

Delete Product

A Product can only be deleted if there are no existing Assets associated with the Product to be deleted. A product needs to be viewed before it can be deleted.

Use Case: AE Maintenance

Scenario: JE Management

A JE is a fixture of Accounting systems. The Journal Entry defines the Debit and Credit account pairs that will be used to post Subledger transactions. The JE does not define how the amount to be posted will be calculated, but defines the account where the amount to be posted will be posted. There will be one debit account and one credit account for every debit credit pair. These will not occur as unpaired accounts.

Role:

Add, update and delete need to be an easily accessible, secured functions available to the AE administrator.

Viewing needs to be available to all AE users.

Frequency / Volume:

During the startup phases there will be many iterations through JE management. Once the system is in production, JE's will be view frequently, but updates will be relatively infrequent.

There will be several hundred JE's.

Input / Output Data:

Each JE can be comprised of one or more debit/credit pairs. The JE will have a JE name and description and a list of debit / credit pairs.

Business JEs:

Every debit credit pair must occur as a debit account and a credit account. It should not possible to create a one-sided sub-ledger transaction entry in this system. Every JE must specify at least one debit / credit pair.

Create New JE

It should be possible to create new JE's for use in the Accounting Engine at any time

- All fields are required except for description
- There needs to be at least one debit and one credit account for each JE.

Viewing a JE

It should be possible to view any JE that exists in the accounting engine including all fields associated with the JE.

It will be necessary to look through a long list including all of the existing JEs looking for a JE without knowing the name of the JE that is needed.

Update a JE

It should be possible to update a JE in the accounting engine. A JE can not be updated until after it has been viewed. This includes deleting debit / credit pairs or adding debit credit pairs.

Delete a JE

It should be possible to delete a JE. A JE can only be deleted if there are no existing Qualified Events that are associated with the JE.

A JE can not be deleted until the details have been viewed.

Issues

Notes

Use Case: Rule Maintenance

Scenario: Rule Maintenance

Rules are used to define the calculations required to process an Event. A Rule is similar in nature to an Excel function. Each Rule will expect specific inputs and will have pre-defined, hard-coded behavior. These are complex in nature and will be built with the expectation that an accomplished spreadsheet user is defining the Rules.

Each Rule can be used in many different Qualified Event lines. E.g. it will be possible to define a Rule for Net Receivable that is referred to in several Qualified Event lines. If there was a different net receivable calculation for which was dependant on Product type, there would be two different Rules, Net Receivable for Product A, and Net Receivable for Product B.

Every JE that is processed will need to use a rule to calculate the posting amount.

Role:

Add, update and delete need to be an easily accessible, secured functions available to the AE administrator.

Viewing needs to be available to all AE users.

Frequency / Volume:

Rules are complex in nature. During the startup phases there will be many iterations through Rule management. Once the system is in production, rules will be viewed frequently, but updates will be relatively infrequent.

Input / Output Data:

The Rule will have header information containing: Rule name, Rule Description

Each Rule will also have one or more Rule Lines. The Rule lines will each have all of the data required to perform the calculation. This may include: verbs (SumStream, SubSL, AddAmt, etc.), operands defining an Entity, Parameters, fixed amounts, Accounting periods, Date, and the destination for the result of this calculation. The final destination for every rule needs to be JEPostAmt.

Valid destinations: User variable, JEPostAmt or StreamPostAmts.

- JEPostAmt is the amount of the JE Debit / Credit entry.
- StreamAmts contain the information required to create or update a stream.
- UserVariables are temporary fields defined in this rule, for use on one or more rule lines.

The data used to process rules can come from:

- ParmList - the parameters that are first created in the Parm Definition scenario, and which will then be selected for use when the Business Event, Product is created. These parms will be passed with the call to the Event Processor.
- Constant - a string, number or percent, which is entered and stored with the Rule when the Rule is created.
- User Variable - a variable defined on a previous line of this Rule.

- AE Field - a field in the AE (or ATLAS) database that has been defined for use in this Rule.

Business Rules:

Rule verbs and expected inputs

The Verb list needs to exist in a look-up table. The table does not need any special GUI support. It can be loaded using SQL since the behavior of each Verb will be captured in code.

Sum an asset stream using a 'from date' and 'to date' to return a currency. The dates will be significant for month and year. This can be used to return the total of an entire stream, a single month, or any number of months.

SumStream: ***Asset, Stream, Begdate, Enddate***

Return: ***Total of stream as currency***

Not needed? Subtract total of work stream from total of current stream (same stream numbers)

DeltaStream streamname begdate enddate assetgeneration1 assetgeneration2

Add dollar amount to sum of stream.

SumStream (as above) – capture sum in variable StreamTotalVariable

AddAmount StreamTotalVariable, Amount

Subtract dollar amount from sum of stream

SumStream (as above)

SubAmount StreamTotalVariable, Amount

Multiply, divide, subtract 2 amounts

MultAmount Amount1,Amount2

DivAmount DivideAmount, byAmount

SubAmount Amount1,Amount2

Pass through amount for posting. This is used to pass an amount in from ATLAS without any special calculations.

PostAmount amount

Sum subledger

GetSLBalance SL# Accounting period – OR –

SumSLGroup SLGroupname Accountingperiod

Add a number to subledger sum

GetSLBalance or ***SumSLGroup*** (as above) -- capture sum in V1

AddAmount V1 amount

Subtract a number from subledger sum

GetSLBalance or ***SumSLGroup*** (as above) -- capture sum in V1

SubAmount V1 amount1

Sum a series of input numbers

AddAmount amount1, amount2).....

Subtract the sum of a subledger group from the sum of another subledger group

SumSLGroup SLGroupname Accountingperiod -- capture sum in V1

SumSLGroup SLGroupname Accountingperiod -- capture sum in V2

SubAmount V1 V2

-- OR --

DeltaSLGroup SLGroup1 SLGroup2 Accountingperiod

Get balance in first debit subledger of a JE (PMS RDR)

RevDebit JE#

Get balance in first credit subledger of a JE (PMS RCR)

RevCredit JE#

Pass through amount for creating or updating a Stream. This is used to pass a Stream from ATLAS without any special calculations.

StreamAmounts StartDate, MonthArray

Valid Dates / Periods

The Date / Period table will be loaded using SQL, not a GUI. This is not expected to change after the initial implementation.

For Stream verbs the following standard dates are valid:

- Current Date: today's date will be used for processing.
- Start Date: the first date in the Stream.
- End Date: the last date in the stream.

For S/L verbs the following dates are valid:

- Current Period: the current fiscal period will be used for processing.
- Previous Period: the previous period will be used for processing.

- Previous Year-End: the 12/31 date of the prior year will be used for processing.

Defaults:

To simplify the entry of Accounting rules when creating Events the following defaults will be used:

Current accounting period

Convention, 0 amounts will not be posted

Beginning to end date range

Create New Rule

It should be possible to create new Rules for use in the Accounting Engine at any time.

- The rule line will only be accepted if it is complete with respect to the fields required for each verb.
- Every Rule will have at least one Rule line.

GUI: Right click on the Rule list to add a new Rule.

Right click on ??? to add a new Rule line.

Populate grid with Rule lines using bound controls for dropdowns to create a new Rule line.

Viewing a Rule

It should be possible to view any Rule that exists in the accounting engine including all fields associated with the Rule. It will be necessary to look through a long list of rules looking for a Rule that has the functionality required to perform a calculation, without knowing the name of the Rule that is needed.

Update a Rule

It should be possible to update a Rule in the accounting engine. A Rule can not be updated until after it has been viewed. This includes deleting Rule lines or adding new Rule lines.

It should be possible to insert in between two existing rule lines.

It should be possible to update any Rule line as long as the last line has a PostAmt destination.

All of the fields that are on the Rule should be available for update.

Delete a Rule

It should be possible to delete a Rule.

A Rule can only be deleted if there are no existing Qualified Events that are associated with the Rule.

If the whole Rule is not being deleted it should be possible to delete any Rule line, as long as there is at least one Rule line and the last line has a PostAmt destination.

A Rule can not be deleted until the details have been viewed.

Issues

Do we have the correct notation for StreamAmt?

Notes

Tables

Verb List: Load with SQL. GetAll method for supporting GUI controls.

Accounting Period List: Load with SQL. GetAll method for supporting GUI controls.

Field List: Maintain with GUI. GetAll method for supporting GUI controls. Get a single Field using the Alias to support Event Processor.

Rule

Rule_Line–

Rule_Var

Rule_Var_Type

Data Notes:

To Post the value of A+B-C = PostAmt

Rule table might look like this:

ID	Name	Action	Destination
1	Rule 1	AddAmount	Variable
2	Dog Rule	SubAmount	PostAmount

Or in English we have resolved the Action and Destination for the following equations:

AddAmount Amt1 + Amt2 = Variable

SubAmount Variable – Amt3 = PostAmount

Now we go to the Rule_Var table to solve for Amt1, Amt2, Amt3 and Variable

SO_Var_ID	Type_Id	Line_Id	Origin Line ID	DBField Id	Parm Id	Constant
1	Parm	1	Null	Null	P123	Null
2	Parm	1	Null	Null	P134	Null
3	Dest	2	1	Null	Null	Null
4	Parm	2	Null	Null	P001	Null

There are 4 Type's that are valid to solve the equation:

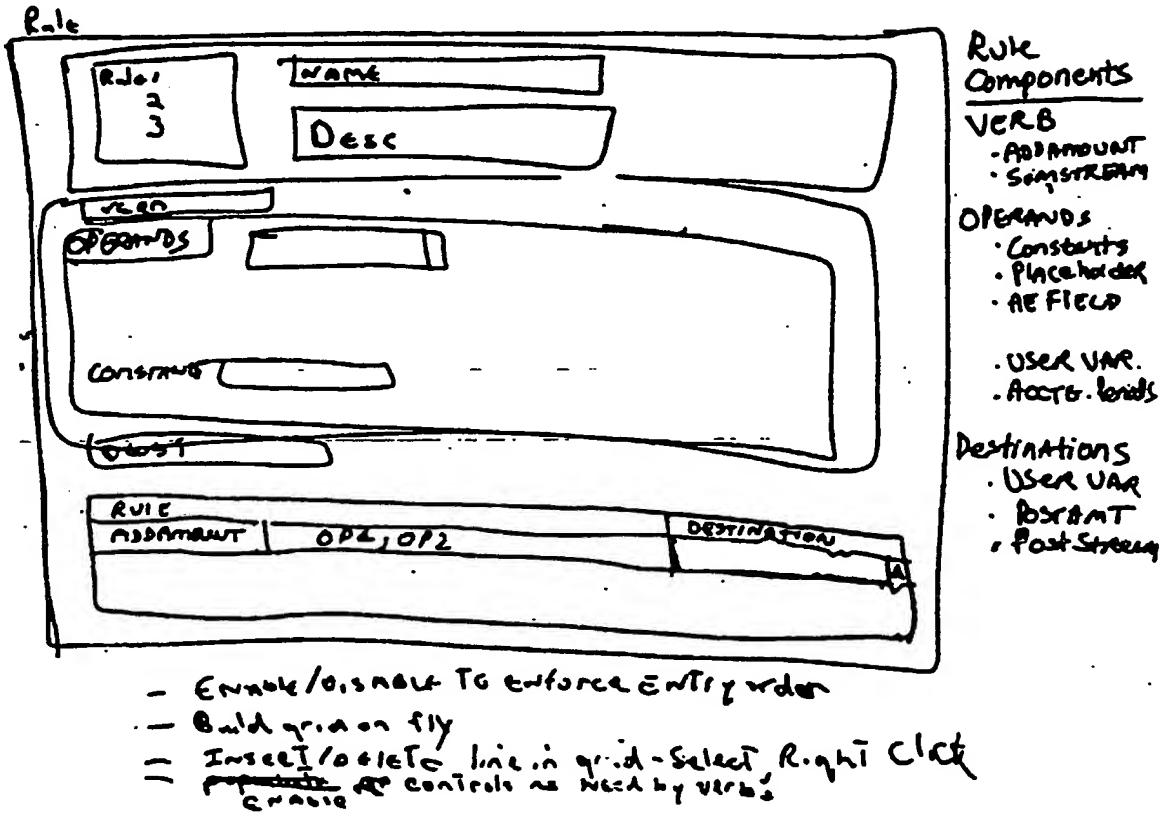
Parm: the data is passed in, in the Event Parm

Dest: a destination (variable) defined on a previous line of the Rule.

DBField: a field on the database

Constant: a numeric. E.g. 1 or .8

Gui Notes:



Accounting Engine Rules

Overview

Accounting Engine Rules are used to describe the calculations performed by the Accounting Engine. Every Business Event will contain one or more rules to calculate each required Journal Entry. Setting up these rules will be approximately as difficult as setting up a set of functions in a spreadsheet. The Accounting Engine team expects this to be consistent with the skill-set of an Accounting or Finance person well versed in Excel or Lotus.

Please note that much of the difficulty in calculating Lease and Loan accounting entries lies within the Pricing or Financial Modelling Tool, SuperTrump. The complexity for the Accounting Engine lies in finding a syntax to use the data created by: SuperTrump, previous accounting entries and the operating system.

Rules

Multiply, divide, subtract 2 amounts

MultAmount amount1(assetgeneration1) amount2(assetgeneration2)

DivAmount dividend amount(assetgeneration)

SubAmount amount1 amount2

Pass through amount for posting

PostAmount amount

Sum subledger

GetSLBalance SL# Accounting period Bookset -- OR --

GetSLGroup SLGroupname Accountingperiod Bookse

Add a number to subledger sum

GetSLBalance or ***SumSLGroup*** (as above) -- capture sum in V1

AddAmount V1 amount1(assetgeneration1)

Subtract a number from subledger sum

GetSLBalance or ***SumSLGroup*** (as above) -- capture sum in V1

SubAmount V1 amount1(assetgeneration1)

Sum a series of input numbers

AddAmount amount1(assetgeneration1) amount2(assetgeneration2).....

Subtract the sum of a subledger group from the sum of another subledger group

SumSLGroup SLGroupname Accountingperiod Bookset -- capture sum in V1

SumSLGroup SLGroupname Accountingperiod Bookset -- capture sum in V2

SubAmount V1 V2

– OR –

DeltaSLGroup SLGroup1 SLGroup2 Accountingperiod Bookset

Implementation of the following verbs has been deferred. It is not clear we will need these verbs.

[redacted]

Defaults:

To simplify the entry of Accounting rules when creating Events the following defaults will be used:

current accounting period

convention that 0 amounts will not be posted

a default bookset

?

Verb Conjunction	Nouns	Variables	Dates	Asset Generation
PostAmount	rent	V1 - V10	begin	current
SumStream	income	AE db fields	end	previous
DeltaStream	interest		billed through	first
AddAmount			.....	work (temp)
SubAmount				
MultAmount				
DivAmount				
SubSLDetail / GetSLBal				
SumSLGroup				
DeltaSLGroup				
RevCredit				
RevDebit				

Use Case: AE Maintenance

Scenario: Event Modifier

The Event Modifier is used to prevent the need for hard-coding special conditions that need to apply to a specific JE. The Event Modifier describes the conditions that must apply before a Qualified Event line evaluates to true.

The Event Modifier will name the exception condition and define how the AE will recognize the data to be evaluated by the Event Processor.

E.g. Fee codes in PMS, may be represented as Event Modifiers.

Role:

The AE administrator enters this information. This information needs to be communicated to I.T. to ensure that the corresponding information is passed from the operational (ATLAS) system.

Any AE user or developer can view this information.

Frequency / Volume:

Definition and Maintenance is primarily a set-up task used by the AE administrator..

The Event Processor will refer to the Event Modifier every time an event is processed. This is a high volume activity.

Input / Output Data:

Every Event modifier will be comprised of one or more event detail lines that will be used when processing this Event Modifier.

Qualified Event Lines will display the Event Modifier by name.

The Event Processor will need to retrieve the Event Modifiers as efficiently as possible. The Event Processor needs to evaluate all lines in the Event Modifier to true before it recognizes the Event Modifier as valid for an Asset. E.g. Event Modifier Sample 1

Source	Field	R. Op.	Value
AE	Principal	>	0
StdParmList	LoanType	=	SBA

In this example the Event Processor will check the principal field (in the AE database) to see if it greater than 0, then it will resolve the StdParmList to find loan type and see if the value is SBA. If both of are true then Event Modifier Sample 1 is true.

Header

Event Modifier Name: will be referred to by ATLAS when invoking the Event Processor, as a selectable field on the Qualified Event window, and used by the Event Processor to evaluate Qualified Event lines.

Event Modifier Description

Detail Lines

The order of the Event Modifier Lines impacts the processing of the Event Modifier. It is important to be able to insert or delete any line and to maintain the order of the lines as they are entered or updated on the Event Modifier Maintenance window.

Event Modifier Line Source: There are two valid sources of data for an Event Modifier line: an AE field or the StdParmList from the Event Product.

Event Modifier Line Field: a selectable control identifying the field in the AE or the parameter on the StdParmList.

Event Modifier Line Relational Operator: =,>,<

Event Modifier Line Value: a constant.

Deleting an Event Modifier

An Event Modifer will delete EMDetailLines and EMHeader information. Referential integrity needs to be enforced to the Qualified Event table.

It should be possible to delete:

- One or more Event Modifier Lines. There needs to be at least one Event Modifer Line for each Event Modifier.
- All of the Event Modifier Lines should be deleted when the Event Modifer is being deleted..

Updating an Event Modifier

Name, Description and the Lines can all be updated.

Adding an Event Modifier

Since many of the Event Modifiers may share similar lines, the Add function should incorporate some concept of clone or copy/paste functionality to speed up a repetitive process.

Notes:

Use the default case rather than an Event Modifier wherever possible. Resolving Event Modifiers for each Asset, Business Event and Product. Event Modifiers should be used when it is necessary to make a special entry that is different than the standard entry for a Business Event and Product.

The Event Modifier is not Event specific. Therefore it is necessary to ensure that any Event Modifier that is using a parameter on the StdParmList, actually has the parm available when we are processing through the Qualified Event Lines. This will be enforced in the Qualified Event Maintenance use case.

Use Case: AE Maintenance

Scenario: Maintaining a Qualified Event

The Qualified Event (QE) is used to guide the processing that will be required of the Event Processor. It will be necessary to define every processing event by associating: Event Modifiers, Earning and Non-Earning Journal Entries, Rules and Booksets prioritized by an Entry Name that will be used in the evaluation of the QE lines.

This scenario is organized around 3 scenarios:

1. Associate Business Event, Product and Parms

2. Maintain Qualified Event Lines.

Scenario: Associate Product Business Event and Parms

The Accounting Engine needs to define and use the list of valid Products and Business Events. This is an association between an existing Product and an existing Business Event. Additionally, each Product Business Event will be able to use one or more Standard Parameters that are passed into to the AE as part of the invocation of the Event Processor.

This scenario describes how to create the association between the Business Event and Product and then how to maintain the Parameters needed for this Business Event Product.

Role:

The AE administrator needs to be able to create the Product Business Event association and to associate the parameters that will be needed to process this event.

Note: the calling system will need to be able to pass in these parameters as part of the call to the Event Processor.

Any AE can view the valid list of Product Business Events and the associated parameters.

Frequency / Volume:

This will be high frequency as part of the AE setup, but will not be updated very often. There will be a few dozen Product Business Events. It is unclear how many parms will actually be required, though it should be less than 250. A typical event will use < 20 parms, though there is no physical limit.

Input / Output Data:

The Product list, the Business Event List, the Standard Parm List and a description of the Business Event Product.

Prerequisites:

Business Event, Product, Rule and Parm all need to exist before the Qualified Event line can be created.

Use Case Product Management

Use Case Business Event Management

Use Case Parm Management

Use Case Rule Management

Business Rules:

It is possible to have 0 parms for the Product Business Event.

The Product Business Event and associated parms will be referenced from outside the AE. Any change to these implies a systems change.

Create Product and Business Event Association

It should be possible to associate any Product in the AE with a Business Event in the AE. It should be possible to add parms to this association, defining the data that will be passed into the AE with the call to the Event Processor.

Since the Event Processor and Qualified Event will both interpret that parameter array by evaluating the position of a parm in the array, it will be necessary to sequence the parameter array to ensure that parameters stay in the correct order.

Remove Association

It should be able to remove the association between any Product and Business Event.

Add Parms

It should be possible to add parms to a Product Business Event at any time. These parms need to be recognized by the system that will call the Event Processor using this Product Business Event

This includes adding a parm at any position of the parm list. This will call for re-sequencing all parameters as required.

Remove Parms

It should be possible to remove a parm from the Product Business Event at any time.

This includes deleting a parm at any position of the parm list. This will call for re-sequencing all parameters as required.

Update Parms

There is no update.

Scenario: Maintaining Qualified Event Lines

After selecting or associating a Business Event Product and creating or updating the Parameter List it will be necessary to maintain the Qualified Event Lines that define this Business Event Product.

This requires the following:

1. Naming the entry (or entries) required.
2. Select an Event Modifier.

3. Select earning JE.
4. Select the bookset(s) in which these entries can be made.
5. Select non-earning JE.
6. Select Rule.

Role:

Qualified Events will be entered by the AE administrator and viewed by all users of the Accounting Engine.

Qualified Events Lines will be evaluated and processed by the Event Processor.

The system calling the Event Processor will need to be able to pass in all of the parameters specified for this Business Event and Product

Frequency / Volume:

The initial creation of Qualified Events will be complex and time consuming. Ongoing maintenance will be limited to the addition of new Products or processes, and will be infrequently performed.

AE system use of Qualified Events is a high volume activity that will be done hundreds or thousands of times a day. This will also be used extensively by special periodic processes like Fiscal close and Year-end.

Input / Output Data:

Qualified Event Maintenance will associate many pieces of existing information. Entry name is the only field that will not have been created by the AE administrator before the Qualified Event is created.

Qualified Event Header

Every Qualified Event is recognized as a unique combination of Business Event and Product, which contains one or more Qualified Event Lines.

Qualified Event Lines

Entry Name: is used to group Qualified Event lines that need to be evaluated to determine which one, and only one, will be processed. This evaluation requires using the Event Modifier to determine which line will be used to create a JE. Lines with a single Entry Name will be evaluated from first to last (a hidden priority), until a line matches on Event Modifier.

Event Modifier: is used to name one or more conditions that need to apply to an asset being processed to recognize this Qualified Event Line as a match. See Event Modifier Scenario to gain a more complete understanding of how the Event Modifiers work. Event Modifier is the only optional field. This is a bound control containing all Event Modifiers.

*** Special note: it is important to validate the list the parameters required by the selected Event Modifier. Event Modifier is not Business Event Product specific. This can result in pointing to an Event Modifier, which uses a parameter that is not included in the StdParmList.

- get selected event modifier.
- check if any parms are required.
- If so, check if parms exist on the Business Event Product Parm list.

- If not, then ask user if this should be added to the Business Event Product Parm list. Append at end, if appropriate.

Earning JE: is the Journal Entry that will be used to create the specific debits and credits required for Earning Assets. This is a bound control containing all of the Journal Entries on the JE table.

Non-Earning JE: is the JE that will be used to create the specific debits and credits required for non-earning assets. If no JE is entered then use the Earning JE for earning and non-earning assets. This is a bound control containing all of the Journal Entries on the JE table.

Rule: is the calculation rule (similar to an Excel function) the will be used to calculate the dollar amount of the debits and credits that will be posted. Rule will also drive the creation of streams, where applicable. This is a bound control containing the list of all Rules. When the Rule is created it will be possible for a Rule to contain parameters, which need to be resolved to a field on the Business Event Product StdParmList at the time the Qualified Event Line is created.

Booksets: are the Booksets that may be used to post this entry. The AE will post to any bookset in this list that applies to the asset. This will enable us to post entries to more than one bookset. It will also enable us to prevent entries from being made in Booksets in which the entries would be inappropriate. These are selected from a bound control containing all valid booksets.

*** This is a modal pop-up that needs to be displayed after the selection of the earning JE.

Business Rules:

The default case for a qualified event line is defined as an empty Event Modifier. If a default case exists it needs to be the last line in an Entry Name group

Every group of Qualified Event Lines in the QE with the same Entry Name will be evaluated to make a single entry.

If there is no match, the default case will be used to make the Journal Entry.

If there is no default case, and there is no match on Event Modifier, then no entry will be made.

Sort by: Business Event, Product, Entry name and priority (a non-display field).

Notes:

The GUI for Qualified Event may be modeled after the Rule GUI. This has not been fully explored.

Sample creation of a single line:

1. Enter Entry Name, free form text, required.
2. Select Event Modifier (see special note above), optional.
3. Select Earning JE, required.
4. Select Non-Earning JE, if applicable.
5. Select Rule, required.
6. Select Booksets, at least one, may be multiples.

Use Case: AE Maintenance

Scenario: Parm Management

Parms are used to define the parameters that will be required to process a Business Event, Event Modifier or Rule. A Parm is a variable name and data type defined to the Accounting Engine and created as a placeholder in the Business Event, Event Modifier and Rule.

This gives us a vehicle for naming and referring to information that will be passed into the AE as part of a call to the Execute Event without hardcoding the signature of every event. Pretty cool.

Role:

Add, update and delete need to be an easily accessible, secured functions available to the AE administrator.

Viewing needs to be available to all AE users and system developers.

Frequency / Volume:

Parms are complex in nature. During the startup phases there will be many iterations through Parm management. Once the system is in production, parms will be viewed frequently, but updates will only be processed when there is an addition or change to an Event call from Atlas to the AE.

Input / Output Data:

Each Parm consists of:

Name: the name of the parameter. This name needs to be easily recognizable so that it can be selected for association with Rules, Business Events and Event Modifiers.

Description: optional

Paramter Type: This identifies the type of data that is being used in this parameter. This can be: String(text or numbers), Numeric(Numbers) or Currency.

Business Rules:

To be defined

Create New Parm

It should be possible to create new Parms for use in the Accounting Engine at any time

- All fields are required, except description.

Viewing a Parm

It should be possible to view any Parm that exists in the accounting engine including all fields associated with the Parm. It will be necessary to look through a long list of parms looking for a Parm that has the functionality required to perform a calculation, without knowing the name of the Parm that is needed.

Update a Parm

It should be possible to update a Parm in the accounting engine. A Parm can not be updated until after it has been viewed. This includes deleting Parm lines or adding new Parm lines..

All of the fields that are on the Parm should be available for update except for the Parm Name.

Delete a Parm

It should be possible to delete a Parm. A Parm can only be deleted if there are no existing Qualified Events that are associated with the Parm.

A Parm can not be deleted until the details have been viewed.

Issues

Notes

Have we described enough data types described. The simpler the better.

Use Case: AE Maintenance

Scenario: Subledger Group Definition

Every subledger group is comprised of one or more subledger accounts from the Chart of Accounts. Subledger groups are used to logically connect individual subledger accounts for reference as a group. This will allow us to sum or display a group of subledger accounts that can be referenced by a single meaningful name. E.g. all income accounts or all receivable accounts.

Subledger group definition encompasses two main functions:

1. Adding, deleting, updating and viewing a Subledger Group.
2. Adding, deleting and viewing the Subledger accounts that comprise a subledger group.

Role:

The AE administrator will modify subledger groups. Any AE user can view the subledger group.

The system will use subledger groups for Rules, Journal Entries, and reporting.

Frequency / Volume:

There is moderate setup to define the Subledger groups. Modification or deletion will be infrequent once the AE is setup correctly. Viewing subledger groups will only happen occasionally.

Input / Output Data:

Subledger group name is required and must be unique.

Subledger group description is required.

A list of subledger accounts in the subledger group.

Business Rules:

A subledger group needs to contain at least one subledger account.

All accounts in the group need to be defined on the Chart of Accounts.

It should be possible to add a subledger group or delete a subledger group that is not being referenced in the AE, add a subledger account to a group at any time, remove a subledger account from a group at any time.

Use Case: AE Maintenance

Scenario: Subledger Definition

Every subledger account needs to be defined in the subledger chart of accounts before it can be used in the Accounting Engine.

Role:

The subledger chart of accounts can be modified by the AE administrator. Any AE user can view the Subledger chart of accounts.

The system will use the subledger chart of accounts with Subledger groups and Journal entries for individual assets.

Frequency / Volume:

There is extensive setup to define the Subledger chart of accounts. Modification or deletion will be infrequent once the AE is setup correctly. Viewing accounts in the subledger chart of accounts will happen almost daily.

Input / Output Data:

Subledger account number

Subledger account name

G/L or Memo indicator

ALER indicator: Asset, Liability, Expense or Revenue.

Indicator to see if this is an active or inactive sl account.

Year-end Roll-up acct. This is the account that will be used for year-end processing to create the 1/1 balance for the new-year.

SL transfer account. This is the account to be used for entries by the office or corp. transfer process.

SL cross-reference. This is a cross-reference (commentary only) field that identifies the subledger that was used on another system.

Business Rules:

A subledger can not be deleted if it is associated with a subledger balance or subledger detail.

Subledger account number needs to be unique.

All fields are required except for sl cross reference.

SL transfer account and year end roll-up account default to the sl account number if they are not entered.

Use Case: AE Maintenance

Scenario: Business Event

Business Events are the 'Events' that will be passed into the Accounting Engine Event Processor. These events are used (along with other information) to determine the appropriate set of accounting entries. Events describe a fairly high level process, such as, Booking, Cash Posting, Billing, Terminations, Loan Disposal, etc.

This is primarily a set-up function designed to create the list of Business Events that will be valid in the AE.

The Business Event will be used in combination with a Product and an Event Modifier to define the Qualified Event. The Qualified Event determines the parameters (data passed into the Event Processor with the Event call) that will be necessary to successfully account for this event and the Journal Entries that will ultimately be created.

Role:

AE administrator can add, update or delete a business event. AE users will have a rare need to view the Business Events.

Frequency / Volume:

There will be < 500 business events. This will primarily be an AE setup activity, with occasional use for systems problem solving.

Input / Output Data:

- Business Event Name is required,
- Description is optional.

Business Rules:

Business Event Name must be unique.

Business Events that are used in calls to the AE are required for successful completion of the desired accounting transaction. If the Business Event does not exist in a Product Business Event association the calling system will be expected to reject the entire transaction to maintain the integrity of the accounting and operational data.

Create

It should be possible to create a Business Event at any time. It will be necessary to communicate the new business event to systems personnel to ensure that calls to the AE recognize this as a valid event.

View

The list of all valid Business Events needs to be viewable as part of the Associate Product, Business Event and Parm use case.

Delete

It should be possible to delete any Business Event that does not have a Product Business Event association.

Update

It should be possible to change the Business Event at any time.

Use Case: Stream Maintenance

Scenario: Streams

The Accounting Engine needs to store streams of data, which represent a series of monthly amounts over time. Accounting Engine Maintenance will define the Name and Description for each stream that needs to exist.

At the current time we expect to use Fees and Expense Streams. This remains to be decided.

Role:

AE administrator will be responsible for defining the Streams that may be used for a Loan. AE users may occasionally view this information.

The AE system will only use this information for display or reporting.

Frequency / Volume:

Stream definition and viewing is very infrequent.

Stream display will be more occasional. There will be very few types of Streams.

Input / Output Data:

Stream Name: required.

Stream Description: optional.

Business Rules:

Streams will need to be defined before they can be passed into the AE from ATLAS or used by the AE in a Rule.

Update a Stream

Any field may be updated.

Delete a Stream

Referential integrity needs to be enforced to the Asset Stream.

Create a Stream

A Stream can be created at any time.

Tables

Asset_Stream_Description

Referential integrity to: Asset Stream.

Issue:

What do we do to ensure referential integrity to rules referring to a Stream that is deleted? This seems like it will need to be programmatic. Perhaps we should eliminate the concept of deleting a Stream type. Let's get real, when are we going to delete a Stream type.

Use Case: Calendar

Scenario: Defining and using the Calendar

The Calendar component is designed as a re-useable component that encapsulates calendar functionality.

This component needs to fill three main needs:

- The calendar needs to capable of defining and mapping a user defined activity to a day on the calendar.
- It should provide the methods required for ATLAS and the Accounting Engine to identify a pre-specified set of dates, relative to any day on the calendar. E.g. previous fiscal year end or day of the week.
- It should provide the fiscal views of the calendar that are required for the Accounting Engine. The calendar needs to be able to:
  - Identify the current fiscal month. This is used for the majority of accounting entries.
  - Identify the fiscal month (and year) for any given day.
  - Identify Calendar month-end, or any other Activity date.
  - Identify the day that fiscal processing should occur for every month in the year.

Role:

The AE administrator maintains the Calendar. The calendar is used extensively by the AE system.

ATLAS will use the calendar component in the billing component.

Frequency / Volume:

The base calendar will be created at the time the system is set-up. There will be very few calendars (<3)

Maintenance for the Accounting Engine should be once a year, to add one more year to the calendar. A year will have a row for every day of the year in the date table.

Calendar maintenance may occur more frequently if the Calendar component is re-used by ATLAS to enhance scheduling, interest calculations, or some other time dependant function.

There will be very few Calendar Activity Types. (<25).

The calendar business services will be referenced for many asynchronous processes. e.g. Billing, accruals, Accounting events.

Related Scenarios:

Defining and maintaining Calendar Activity Types.

Calendar Activity Dates.

Getting dates relative to a date.

Input / Output Data:

There are three basic pieces of information that make a calendar meaningful for the accounting engine:

- Calendar header information: Name and Description that can be used to identify which calendar is being used by an Asset or Bookset. Last update date, last updated by, current fiscal month and current fiscal year are also stored with the calendar.
- Calendar days: Physical day and fiscal month. It should be possible to retrieve information for every day of the year.
- Calendar Activity Type: a unique Activity Name and any Description. E.g. Calendar month end, fiscal month end. A special reserved indicator will be created to indicate that some activity types need to exist for the AE to successfully process entries and can not be updated or deleted. Activity Type Name and description is required.

Creating the Calendar

A base calendar containing the Calendar name and description needs to be created once for every calendar that will be used in the calendar component.

It will be necessary to add one year of days, in full year increments, to the calendar. At the time the calendar days are created, the user will specify the fiscal month, by starting date of month, for the entire year. This should be implemented as a simple, easy to understand GUI. Since this is done once a year, it is important that this is intuitive and easy to use. A reasonable processing delay (1 minute) is acceptable.

The system should update the user-id and update date.

The accrual process will need to change the current fiscal month and year.

Deleting the Calendar

It will not be possible to delete an entire calendar. This seems both complex and unlikely to be used.

It should be possible to delete (purge) a range of dates as part of standard maintenance.

It should be possible to delete the association between a calendar activity and a day.

Updating the Calendar

It is possible to update the calendar activity type name and description. The system should update the user-id and update date. The fiscal accrual process should update the fiscal month and year.

It is possible to update the days in a calendar by changing the fiscal start month for one or more periods. This will need to change every day in the year to reflect the current fiscal period.

The first day of each fiscal year should be 1/1.

Each fiscal period must have a fiscal start date that is less than the date of the following fiscal period. E.g. If fiscal March starts on 2/28/2000, then fiscal February must start after 1/1/2000 and before 2/28/2000.

Viewing the Calendar

It should be possible to view the following information for a calendar:

- A list of all the Calendars.
- A list of all Activity Types.
- The fiscal date for any day in the calendar.
- A calendar representation of the activities, by day.

Business Rules:

It is necessary to associate an asset with a single calendar.

It should be possible to implement multiple calendars within the Calendar component.

Tables: Calendar_Date, Calendar_Activity_Type, Calendar_Activity_Date

Scenario: Defining and maintaining Calendar Activity Types

The Calendar will need the ability to associate activities with dates. The first step in creating this association will be the creation of the Calendar Activity Types.

The second step will be associating the dates in the Calendar with Activity Types that have already been created and will be documented in the Calendar Activity Dates scenario.

Input / Output Data:

- Calendar Activity Type Name and Description are the only two fields that can be created and maintained for the Calendar Activity Type.
- Last Update, Updated by and Reserved Indicator will be generated by the system. The system will always set the Reserved Indicator off.
    - Reserved Activity Types need to be created by I.T. personnel.

Creating Calendar Activity Types

Calendar Activity Types need to be created once for use by all calendars that will be defined in the calendar component.

Calendar Activity Types will not be viewed or changed very often. The most frequent use of Calendar Activity Types will be viewing the Calendar Activity Type associated with a given day.

Calendar Activities can be 'reserved' for system use by setting an indicator on the Calendar Activity Type table. This can only be done by an I.T. system administrator role using SQL or database tools.

Calendar Activity Type Name needs to be unique.

~~The system should update the user id and update date.~~

Deleting Calendar Activity Types

It will be possible to delete a Calendar Activity Type that is not associated with any Activity Date. Attempting to delete a Calendar Activity Type that is used by an existing Calendar Activity Date will cause referential integrity error.

It is not possible to delete a Reserved Activity Type.

Updating Calendar Activity Types

It will be possible to update Calendar Activity Type Name and Description for activities that are not reserved.

It is not possible to update a Reserved Activity Type.

Viewing Calendar Activity Types

It should be possible to view the following information for a calendar:

- A list of all Activity Types: Name and Description.
- A list of all Reserved Activity Types: Name and Description.

Tables: Calendar_Activity_Type, Calendar_Activity_Date

Scenario: Calendar Activity Dates

The Calendar component will need the ability to associate activities with dates in a single Calendar. This scenario assumes that a Calendar has been created and that Calendar Activity Types have been created.

The process is fairly simple.

1. Select the Calendar that will use the Activity Dates.
2. Select the Fiscal Year to which the Activities Dates are being applied.
3. Select the day or Activity date.
4. Add, Delete, or Update an Activity Date.

Input / Output Data:

- Calendar.
- Fiscal Year.
- Calendar Activity Date and Activity Type.

Creating Calendar Activities

Calendar Activity Dates are the junction of a day on a Calendar and an Activity Type. Selecting an existing date on the Calendar, and an existing Activity Type creates the Calendar Activity Date. The UI should display the date, activity type and activity type description for easy viewing.

Many activities can occur on a single day.

The system should update the user-id and update date.

Deleting Calendar Activities

It will be possible to delete any existing Calendar Activity Date. There is no referential integrity to enforce.

Updating Calendar Activities

It will be possible to change the Calendar Activity Type associated with a date, or to change the date for any activity by deleting and re-adding the activity type

Viewing Calendar Activities

It should be possible to view the following information for Calendar Activities on each Calendar:

- Activity Date, Activity Type and Activity Description.

Tables: Calendar_Activity_Type, Calendar_Activity_Date, Fiscal_Calendar

Scenario: Getting dates relative to a date

The Calendar component will be required to provide the methods required for ATLAS and the Accounting Engine to identify a pre-specified set of dates, relative to any day of the year. This is a system requirement, there is no User Interface required.

Input / Output Data:

Input:

- Date, optional, byref. This will default to today's date if it is not entered.
  - Note: there will be times when the system (ATLAS or AE) will need data based on another point in time. By allowing the date as an input field we ensure that we can create the date view for any point in time.
  - Note: this will also ensure that date errors are not generated due to time differences on the servers used to run various applications or components.

Outputs:

Every input request that is successfully processed needs to generate all of the following outputs relative to the input date.

- Today's date: the calendar date for today as obtained from the Calendar application server.
- Input date: this is returning the date that was passed in the input request. This will default to today if the date is not passed as an input argument.
- Day number: The day of the month for the input date.
- Day of the Week: Mon., Tues., etc. expressed as an integer.
  - 1 Sunday, 2 Monday, 3 Tuesday, 4 Wednesday, 5 Thursday, 6 Friday, 7 Saturday
- Day of the Week #: This is used to identify whether the day of the week is the 1st, 2nd, 3rd, 4th or 5th occurrence in the month.
- Last occurrence in month: Boolean. This will be true if this is the last time this day of the week will occur in this month.
- Two weeks ago: The input date minus 14 days, returned as a full date.
- One month ago: The input date minus one month. If the day referenced in the input date is greater than the highest day in the previous month, then this will return the highest day in the previous month. e.g. if the input date is 2/27/99, the return date will be 1/27/99. If the input date is 12/31/99, the return date will be 11/30/99.
- Two Months ago: The input date minus two months. If the day referenced in the input date is greater than the highest day in the month before the previous month, then this will return the highest day in the month before the previous month. e.g. if the input date is 2/27/99, the return date will be 12/27/98. If the input date is 08/31/99, the return date will be 06/30/99.

Tables: None

Scenario: Getting fiscal dates relative to a date

The Calendar component will be required to provide the methods required for ATLAS and the Accounting Engine to identify a pre-specified set of fiscal dates, relative to any day of the year. This is a system requirement, there is no User Interface required.

Input / Output Data:

Input:

- Calendar Entity Id, required.
- Date, optional, byref. This will default to today's date if it is not entered.
  - Note: there will be times when the system (ATLAS or AE) will need data based on another point in time. By allowing the date as an input field we ensure that we can create the date view for any point in time.
  - Note: this will also ensure that date errors are not generated due to time differences on the servers used to run various applications or components.

Outputs:

Every input request that is successfully processed needs to generate the following outputs relative to the input date. This will be done in an Enum.

- Input date: this is returning the date that was passed in the input request. This will default to today if the date is not passed as an input argument.
- Fiscal Period: the fiscal period for the input date expressed as a month and year.
- Fiscal month End: the date of the fiscal month end for the input date. This is calculated as the day before the start of the next fiscal month.
- Previous Fiscal Period: the (fiscal period – 1) for the input date. If the fiscal period for the input date is period 12/1998, then this will contain the value 11/1998. If the fiscal period for the input date is period 01/1999, then this will contain the value 12/1998.
- Previous Fiscal Year end: The previous fiscal year end relative to the input date. If the fiscal period for the input date is period 04/1998, then this will contain the value 12/1997. If the fiscal period for the input date is period 01/1999, then this will contain the value 12/1998.

Tables: Fiscal_Calendar, Calendar_Date, Calendar_activity_type, Calendar_activity_type_date.

See code in IeventProc DetermineActivityPeriod

Use Case: UDF

Scenario: Maintain UDF Name and Description

User Defined Fields (UDF's) can be created in the UDF component for reporting and inquiry. UDF's will be created to improve the flexibility of the Accounting Engine. This will allow storing data that is not core accounting data, but which is useful in combination with the accounting data. This will reduce the amount of maintenance required in the AE to accommodate special cases, which require non-financial data.

It is necessary to perform standard maintenance on a User Defined Field to Add, Update, and Delete the UDF Name, Description and Table Name.

Every UDF will identify one table on the database, which will be used to resolve the UDF to a specific Asset, Asset Group or other instance on an existing table. The table name will be stored with the UDF Name and description to ensure that all instances used by a single UDF Name are resolved using the same table.

UDF Maintenance will be performed by a call to the AE from the ATLAS system.

UDF's should be carefully implemented to prevent the proliferation of UDF's in the AE. This is a relatively inefficient structure. It should be used to enhance flexibility, not as an extensive reporting aid.

Role:

ATLAS system will be the actor in the Maintain UDF use case.

Frequency / Volume:

This maintenance will be rare. There will only be a handful of UDF's used in the initial SBF implementation of the AE.

Input / Output Data:

User Defined Field Name: required and unique.

User Defined Field Description: optional.

Table Name: required. This needs to identify an existing table accessible by the AE.

Business Rules:

Do not delete UDF Name that is being referenced by a UDF name/ value pair.

Tables:

UDF_Names: It should be possible to retrieve the entire list of valid names.

Scenario: Maintain UDF Values

It is necessary to perform standard maintenance on a User Defined Field to Add, Update, and Delete the UDF Name and Values Pairs. This is the process of using UDF data with UDF's that have been maintained in the *Maintain UDF Name and Description Scenario*.

UDF Maintenance will be performed by a call to the AE from the ATLAS system.

Role:

ATLAS system is the actor, which maintains the UDF values.

Frequency / Volume:

Frequency and volume are unpredictable at this time. We are aware that this is a relatively inefficient process that can easily be abused. This should be monitored and updated, as required.

Input / Output Data:

The UDF Name / Value pair is basically comprised of three types of data: the name of the UDF, the value of the UDF and Table and InstanceId for the Entity that is associated with the UDF Name and Value pair. E.g. UDF Name: *Account Schedule Salesperson*, UDF Value: *Salesperson Jane Doe*, for Asset Group: *AS1234*.

User Defined Field Name: required and unique.

Value: data string

InstanceId: the Id of the entity from the AE. This is used to associate the value with a specific Asset, Asset Group or other entity. Note:This will resolve by accessing the UDF Name table, finding the table to which this InstanceId refers, and using the InstanceID as the EntityId for that table.

Business Rules:

Field Name needs to exist in UDF.

Data will be stored as string.

InstanceId needs to exist at the time of creation, on the table named in the UDFName . This is not necessarily enforced at the DB level.

Tables:

UDF_Names, UDF_Values

Samples:

Header that was set-up in maintenance case.

UDFName	UDF Description	Table
Salesman	salesman # for loan	Asset Group
BDE Loan #	loan program #	Asset

UDF Values

UDF Name	UDF Value	InstanceId
Salesman	Jane Doe	AG 1234
Salesman	John Doe	AG 3456

Use Case: Financial Organization

Scenario: Maintaining the Financial Organization

The financial organization is used for the General Ledger and for reporting. It must be possible to maintain the corporations, businesses (regions), and offices (territories, cost centers) used in the accounting engine.

It should also be possible to query the component for available financial organization entities: Corporation and Office.

Role:

The AE administrator role has the authority to add, delete or update any entity in the Financial Organization.

The AE and ATLAS user roles can view all of the Corporations, Businesses and Offices defined in the AE.

Frequency / Volume:

SBF currently has 2 corps, 2 businesses, and about 2 dozen offices. No growth is expected in the number of corps or business. The number of offices may triple over time.

Maintenance will be infrequent after the initial set-up of the AE.

These tables will be used frequently for reporting.

Input / Output Data:

Corporation:

Name: Required text.

Description: Optional text.

Fiscal Calendar: Every corporation needs to use one and only one fiscal calendar. Required selection from a list of calendars defined to the AE.

Office:

Name: Required text.

Description: Optional text.

Business: Required selection from a list of business defined to the AE.

Business:

Name: Required text.

Description: Optional text.

Business Rules:

- Every office will belong to a single business
- Every business may contain multiple offices

- Every office may be in multiple corporations
- Every corporation may have multiple offices

Scenario: Using the Financial Organization

Every loan is associated with a corporation, a business and an office. This association is maintained for the life of the loan to ensure that loan accounting and reporting are both correct with respect to the financial organization. Any change in the Office or Corporation associated with a loan implies a set of *transfer* accounting entries. A change in the business does not impact the underlying accounting.

Role:

ATLAS user role will pass the association between a loan (asset) and the corporation and office that will be used for accounting purposes when an asset is created.

? where will an office transfer be originated?

Frequency / Volume:

These are transactions scattered throughout the day. In the event of a portfolio acquisition there may be batch processes used to load or create hundreds of these in a short period of time.

Input / Output Data:

The association of the Loan, Corporation and Office are best defined as part of the interface document used to define how to create an asset in the Accounting Engine. At a minimum the AE will require the following inputs:

Loan number, Corporation, Office and an external asset reference number.

The accounting engine will be required to return an asset number to ATLAS to identify the accounting asset.

Business Rules:

Any change in the corporation or office associated with the loan (financial asset) requires office transfer or corp. transfer accounting entries. Maintenance of office and corporation in ATLAS and the AE must be treated as a tightly coordinated process.

Use Case: Subledgers

Scenario: Subledger Transaction Process

Every asset will have subledgers that are comprised of a yearly series of month ending balances for various accounts in the Chart of Accounts. Each subledger will also have a complete transaction detail to support these balances.

Role:

The creation and update of Subledgers is controlled by the AE Event Processor. Subledgers can be viewed by any Accounting User.

Frequency / Volume:

This is a high volume activity. There will be an update to the subledger balance and the creation of a subledger transaction detail for every single ATLAS event that has financial impact. This includes: A/P, Cash Posting, Booking and Terminations.

Input / Output Data:

Information required to create the G/L, details of the financial transaction, 14 (or 15) monthly balances as follows: the balance as of 1/1, 12 month ending balances, Balance of activity to date.

The concept of a "Jan next" needs to be evaluated.

Business Rules:

Think audit.

1. Every debit must have a matching credit. These will always be created in offsetting pairs.
2. Every balance update must have the supporting debits and credits.
3. Debits and credits should be tightly controlled and created in a single easily identified place to prevent problems and provide easy audit of the process.
4. Every row added or updated should have a complete audit trail, who, what, and when.
5. Transaction detail is not deleted unless all activity for a Asset is also deleted.
6. Transaction balances can be updated, but transaction detail will never be changed.
7. Only post to the current fiscal processing month. Do not post to a prior (closed) or future month.
8. Post for an asset.
9. Bank is only used for Cash receipt transactions.

Actions

It is necessary to Post subledger transactions for an asset, for any subledger that is valid in the chart of accounts.

It is necessary to identify the month ending balance associated with an asset for every month in the current year.

It should also be possible to answer the following questions related to subledgers:

1. What is the subledger balance, for a single asset, for a specific month.
2. What is the sum of the balances for a subledger account, for all of the assets in an asset group, for a specific month.
3. What are the month ending balances, by month, for all of the assets in a specific year.
4. What are the Subledger transaction details for a given period of time for a Subledger account.
5. What Subledger balances exist for an asset for a given accounting period.

Issues

1. We have deferred the implementation of using Asset Groups for Subledgers.

Use Case: Subledgers

Scenario: Subledger Group, Subledger Transaction Processing

A Subledger Group is a group of logically related Subledgers that have semantic meaning to the business. E.g. the list of income subledgers might be called the IncomeSubledgerGroup, and the list of receivable subledgers might be referred to as the ReceivableSubledgerGroup. This allows the group to be acted or reported upon in concert, without having to specifically name each subledger to perform the action or create the report.

Role:

The system role can see or refer to any valid subledger group.

Frequency / Volume:

Subledger groups will be referenced frequently by the Event Processor, inquiries and reporting.

Estimate between 10 and 250 Subledger groups. A typical subledger group will contain about 5 – 10 subledgers. It is unlikely that a subledger group will have more than 20 subledgers. It must have at least one subledger.

Input / Output Data:

Subledger group name, description, subledgers associated with the subledger group.

A subledger can be associated with many Subledger groups.

Business Rules:

Every Subledger group will include at least on subledger account.

Actions

It should be possible to retrieve information about subledger groups to answer all of the following questions:

1. What is the sum of the balances for the subledger accounts in a subledger group, for a single asset, for a specific month.
2. What is the sum of the balances for the subledger accounts in a subledger group, for all of the assets in an asset group, for a specific month.
3. What is the sum of the month ending balances, by month, for all of the assets in a subledger group for a specific year.
4. What are the Subledger transaction details for a given period of time for every Subledger account in the Subledger group.
5. What Subledger balances exist for an asset for a given accounting period.

Use Case: Asset

Scenario: Use an Asset

An Asset can represent a physical piece of equipment or a financial entity such as a loan or an unapplied cash account. All Assets will have a corresponding asset represented on the source (ATLAS) system. The ATLAS representation may be a loan or it may be suspense account.

Since the Asset does not contain any financial information the accounting engine will only store the current representation of the Asset. All of the financial information for an asset will be stored in Subledgers or Streams.

Role:

System actors, not human actors, use assets. ATLAS and the AE will both need to use Assets.

Frequency / Volume:

There will be a low to moderate volume activity against the Asset. It will be referenced primarily at the time of booking, for Subledger inquires and reporting.

Input / Output Data:

The asset contains the following:

Office / Corp : Office and Corp need to be validated as a valid Office / Corp. combination.

Facility: the facility (external system, ATLAS) that combined with the external asset reference will be unique.

External asset reference: This is the way ATLAS refers to this asset. This is an important cross-reference point for ensuring the integrity of ATLAS to the AE.

Currency: US $ will be used for SBF implementation.

Permanent asset id: this is used to ensure that we can refer back to the original asset in the event it is necessary to book this deal with a new Loan number.

Volume Ind: is this account new volume for this month? Should this be a volume date?

Status date: date of the last status update.

Asset status: Active, Inactive, Pre-book.

Earning status date: date of the last earning status update.

Earning Status: Earning, Non-earning.

Business Rules:

It is not possible to physically delete an Asset. Asset will be logically deleted by marking them inactive and populating the Status date.

Actions

From the AE or from ATLAS it should be possible to view:

- All of the asset detail for the asset.
- The subledger detail for the asset for a date range.
- A subledger balances for the asset for a period.
- The sum of the subledger balances for all of the subledgers in a subledger group for the asset for a given period.

Use Case: Asset Group

Scenario: Use an Asset Group

An Asset Group is used to identify multiple assets as related entities. This relationship can define things as diverse as assets on an Account, Customer or Vendor. It will be possible to inquire or report against various pieces of data in the AE by specifying the Asset Group. E.g. The AE will be able to return the subledger balance for an asset group by summing a single subledger account for all assets in the asset group.

Role:

System actors, not human actors, use assets groups. ATLAS and the AE will both need to use Assets groups. This will be very useful for reporting and online inquiries.

Frequency / Volume:

There will be a low to moderate volume activity against the Asset group. It will be referenced primarily for Customer inquiries, Subledger inquires and reporting.

Input / Output Data:

The asset group contains the following:

Group type: an identifier to specify why these assets are grouped together. Thus it will be possible to have one group which is used to represent a customer, and another for a portfolio being serviced.

External Group Reference: the reference (name, description or id) the external system (ATLAS) uses to refer to this asset.

Business Rules:

Actions:

From the external (ATLAS) system it should be possible to:

- Add a new asset group.
- Delete an asset group.
- Add assets to an existing asset group
- Remove assets from an existing asset group.
- It will not be possible to update an asset group.

From the AE or from ATLAS it should be possible to view:

- All of the asset detail for the assets associated with an asset group.
- The subledger detail for all of the assets in the asset group for a date range.

- The sum of the subledger balances for all assets in an asset group for one single subledger for a period.
- The sum of the subledger balances for all of the subledgers in a subledger group for all of the assets in an asset group for a period.

Use Case: Processing an Event

Scenario: ExecuteAsset/ExecuteAssetGroup

The Event Processor will be used to post all debits and credits in the Accounting Engine. The Event Processor will also be responsible for maintaining any streams that are used in the AE.

The Event Processor will have a public interface that can be invoked by ATLAS, or an AE internal process. Each invocation will identify the Business Event being processed and pass in any information required to correctly process the Event.

The Event Processor will iterate through the Qualified Event Lines for a Business Product Event, identifying the required Journal Entries by resolving the Event Modifiers for each line, and then processing the Rules required to perform the calculations needed to successfully create the posting amount.

The Event Processor can be invoked to process an Event for a single Asset or for an Asset Group.

Every successful ExecuteAssetEvent will return a unique transaction id to the invoking system unless the invoking system passes in the transaction id of a previous transaction to ensure they look like a single transaction in the AE.

Role:

This is a system actor. There is no direct human Actor.

Frequency / Volume:

This is used for every single debit and credit posted in the AE. This is high volume, time critical process.

Input / Output Data:

Business Event

Product

Asset ID

Unique Transaction ID

Business Product Event standard parameters

Business Rules:

See Rules for Verb list.

Processing

Technical look:

ExecuteAsset (byval astrEvent : string, byval alProductId? : long, byval Assetid : long, byref alTransID : long, byval StdParmList : VariantArray)

ExecuteAssetGroup (byval astrEvent : string, byval alProductId? : long, byval alAsseGrouptid : long, byref alTransID : long, byval StdParmList : VariantArray)

Validity checks

- Lookup / Verify the Business Event – Product combination. This needs to be validated in two places:
  - The Business Event – Product table which is used to resolve the StdPArmList
  - The Qualified Event table to ensure that there is at least one Qualified Event Line for this Business Event – Product.
- Verify the Parms passed in the Standard Parm list (VariantArray) against the parameters expected by the Business Event Product.
  - Have the correct number of parms been passed in to the AE. The number of parms must = the UpperBounds of the EventProdParm.
  - Are the parms of the correct type: currency, string, etc
- Verify Business Event Product Parms against the parameters expected by the Event Modifier.
  - Event Modifer exists
  - Parms are present and of the correct type.
- Resolve Bookset to Asset Bookset to identify the Qualified Event Lines that are using booksets that may be used to make Journal Entries for this Asset. It is not necessary to process QE Lines that do not have a Booksets valid for this Asset.
- Check validity of the Asset being used to process this event.

Qualified Event Processing

- Obtain Qualified Event Lines.
  - Sort by Entry Name and Priority.

For Each QE Line, until a match has been found for this Entry Name:

- IF line has an Event Modifer
  - Has this Event Modifier been checked already? If so, then use the results of the last check to save processing time, if not Check Event Modifer conditions
    - Save the results of the Event Modifier evaluation so we know if the EM is True, False or not evaluated yet.
    - IF they don't match, skip to next line

Process one Qualified Event Line

- Grab Rule
  - Execute SQL select(s) to obtain ?? data (or obtain info from existing RS. E.g. Asset_AE to get Principal Balance.
  - Exec ute Rule using:
    - StdParmList

- EMParmList
- User Variables
- Constants
- AE Fields
- Verbs: AddAmt, SumStream, etc.

- Repeat until Dest = PostAmount

Post the JE

- Obtain JE and supporting tables for S/L Balance
- Process JE (for each JE line) and detail
- Create Subledger Detail for the above S/L

APPENDIX C
FORM INTERFACE DEFINITIONS

LOGICAL VIEW REPORT

SELECTED LOGICAL VIEW REPORT

Bookset
FrmUDFMaint

Private Attributes:

cFORM_MIN_HEIGHT : = 2505
cFORM_MIN_WIDTH : = 7395
gbUpdateUDF : Boolean
gbAddNewUDF : Boolean

Public Operations:

newData 0 :
deleteData 0 :
cancelUDF 0 :

=

METHOD : cancelUDF
PURPOSE : This will cancel an add to the udf Record set
PARMS :

RETURN :

=

deleteUDF 0 :

=

METHOD : deleteUDF
PURPOSE : This will delete a record from the udf Table
PARMS :

RETURN :

=

missingData 0 : Boolean

=

METHOD : missingData
PURPOSE : This method will determine if any required fields are missing
PARMS :

RETURN : lbBad As Boolean

=

saveUDF 0 :

=

METHOD : saveUDF
PURPOSE : This will determine if the data should inserted or updated.
The appropriate call to the business service will be made
PARMS :

LOGICAL VIEW REPORT

RETURN :

=

newUDF 0 :

=

METHOD : newUDF
PURPOSE : this will create a new record on the UDF recordset
The method will also set the pop up menus to their appropriate settings
PARMS :

RETURN :

=

disableText 0 :
enableText 0 :
setTextFields 0 :

=

METHOD : setTextFields
PURPOSE : This method will bind the text fields and drop downs to the data
PARMS :

RETURN :

=

getAllUDF 0 :

Private Operations:

Form_Unload (Cancel : Integer) :
sndPanelMsg (amsgType : panelMsg) :

=

METHOD : sndPanelMsg
PURPOSE : This will display the appropriate message to the panel
PARMS :
amsgType [panelMsg] = the type of message that should be displayed
RETURN : None

=

Form_Load 0 :

=

METHOD : Form_Load
PURPOSE : This event will retrieve the appropriate data, build the grid
PARMS :

RETURN :

=

LOGICAL VIEW REPORT

dgrdUDF_RowColChange (LastRow : Variant, LastCol : Integer) :
dgrdUDF_PostEvent (MsgId : Integer) :
dgrdUDF_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :
dgrdUDF_BeforeRowColChange (Cancel : Integer) :
sbarUDF_Click (Tool : ActiveBarLibraryCtl.Tool) :

MDImain
Product
frmBusinessAdd

Public Operations:

missingData () : Boolean

Private Operations:

Form_Unload (Cancel : Integer) :
Form_Load () :

frmBusinessMaint

Private Attributes:

cFORM_MIN_HEIGHT : = 3555
cFORM_MIN_WIDTH : = 8670
gbUpdateBusiness : Boolean
gbAddNewBusiness : Boolean

Public Operations:

deleteData () :
addNewBusiness () :
newData () :

Private Operations:

disableText () :

==========
=
METHOD : disableText
PURPOSE : This will disable the text entry areas
PARMS :

RETURN : None
==========
=

enableText () :

==========
=
METHOD : enableText
PURPOSE : This will enable all the fields
PARMS :

LOGICAL VIEW REPORT

RETURN :

=

cancelBusiness 0 :

PARMS :

RETURN : None

=

sndPanelMsg (amsgType : panelMsg) :

=

METHOD : sndPanelMsg
PURPOSE : This will display the appropriate message to the panel
PARMS :
amsgType [panelMsg] = the type of message that should be displayed
RETURN : None

=

newBusiness 0 :

PARMS :

RETURN : None

=

checkForMissingData 0 : Boolean

=

METHOD : checkForcheckForMissingData
PURPOSE : This method will determine if any required fields are missing
PARMS :

RETURN : lbBad As Boolean

=

deleteBusiness 0 :

=

METHOD : deleteBusiness
PURPOSE : This method will delete the requested data from the data base.
PARMS :

RETURN :

=

Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :

=

METHOD : Form_QueryUnload
PURPOSE : This method will determine if there is any data missing and then determine if the data should be saved. If data is missing the appropriate message will be sent.
PARMS :

RETURN :

=

LOGICAL VIEW REPORT

saveBusiness 0 :

=

METHOD : saveBusiness
PURPOSE : This method will determine if it should create or update the record.
The method will then call the appropriate method and request the
service from the Business Service Layer
PARMS :

RETURN :

=

setTextFields 0 :

=

METHOD : setTextFields
PURPOSE : This method will bind the database fields to the text fields
PARMS :

RETURN :

=

getAllBusinessData 0 :

=

METHOD : getAllBusinessData
PURPOSE : This event will retrieve all the Business data.
PARMS :

RETURN :

=

Form_Load 0 :
dgrdBusines_RowColChange (LastRow : Variant, LastCol : Integer) :

=

METHOD : dgrdBusines_RowColChange
PURPOSE : This event will post an event if the prior row was saved
PARMS :

RETURN :

=

dgrdBusines_PostEvent (MsgId : Integer) :

=

METHOD : dgrdBusines_PostEvent
PURPOSE : This event will be triggered after the PostMsg. It is here where the
grid will be refreshed.
PARMS :

RETURN :

=

dgrdBusines_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :

=

LOGICAL VIEW REPORT

METHOD : dgrdBusines_MouseDown
PURPOSE : This event will determine if the right mouse button was pressed, if so the active bar for that grid will be displayed
PARMS :

RETURN :

---

dgrdBusines_BeforeRowColChange (Cancel : Integer) :

METHOD : dgrdBusines_BeforeRowColChange
PURPOSE : The BeforeRowColChange will determine if there is missing data and determine if data has been changed. If so the data will be saved
PARMS :

RETURN :

---

abarBusiness_Click (Tool : ActiveBarLibraryCtl.Tool) :

METHOD : abarBusiness_Click
PURPOSE : This method will determine what option was selected within the active bar.
PARMS :

RETURN :

---

frmCalendar

MODULE : frmCalendar
PURPOSE : This form is used to:
1. View, select and maintain calendars
3. view, add and delete activity type dates

Private Attributes:

cActivityTypeFiscalStart : = 1
cFORM_MIN_HEIGHT : = 8355
cFORM_MIN_WIDTH : = 9735
Some Form Constants
gbRefreshActivity : Boolean
gbUpdateActivity : Boolean
gbRefreshCalendar : Boolean
gbUpdateCalendar : Boolean
gstrCalendarName : String
glCalendarID : Long

LOGICAL VIEW REPORT

Public Operations:

resetAllControlsInError 0 :

=

METHOD : resetAllControlsInError
PURPOSE : This method will reset the controls
PARMS :

RETURN : None

=

newData 0 :

PARMS :

RETURN : None

=

Private Operations:

validateActivity 0 : Boolean

=

METHOD : validateActivity
PURPOSE : this method will validate the data that was entered
PARMS :

RETURN : Boolean

=

sndPanelMsg (amsgType : panelMsg) :

=

METHOD : sndPanelMsg
PURPOSE : This will send the appropriate message to the panels status
PARMS :
amsgType [panelMsg] =
RETURN : None

=

ValidateForm (aAction : frmCalendarRSActions) : Boolean

=

METHOD : ValidateForm
PURPOSE : check each recordset that can be updated to see if it has changed.
if it has changed, then validate the data entered
if validations are passed, then call the appropriate save routine
PARMS :
aAction [frmCalendarRSActions] =
RETURN : Long

=

processRS_Calendar (aAction : frmCalendarRSActions, avParms() : Variant) : Long

=

METHOD : processRS_Calendar

LOGICAL VIEW REPORT

PURPOSE : This is the brains of the operation. When it is called it:
1. checks the action used to call it
2. then it enables and disbles controls
3. checks recordsets to see if they need to be validated and saved
PARMS :
aAction [frmCalendarRSActions] =
avParms [Variant] =
RETURN : Long

SaveCalendars 0 : Variant

METHOD : SaveCalendars
PURPOSE : This function will save any changes made to the Calendar grid.
PARMS :

RETURN : Variant

SaveActivityDates 0 : Variant

METHOD : SaveActivityDates
PURPOSE : This function will save any changes made to the activity type / date grid
PARMS :

RETURN : Variant

UnlockActivities 0 :

METHOD : UnlockActivities
PURPOSE : Unlock and enable the activities dropdowns.
PARMS :

RETURN : None

LockActivities 0 :

METHOD : LockActivities
PURPOSE :
PARMS :

RETURN : None

PopulateActivityDates 0 :

METHOD : PopulateActivityDates
PURPOSE : Use the year selected to retrieve all 365/366 days
in the current year.
PARMS :

LOGICAL VIEW REPORT

RETURN : None

PopulateActivityTypes 0 :

METHOD : PopulateActivityTypes
PURPOSE : load the activity drop down used to select an activity type to associate with a date.
this dropdown is only used to add a new activity / activity date association
PARMS :

RETURN : None

PopulateCalendars 0 :

METHOD : PopulateCalendars
PURPOSE : load the calendar grid with all the calendars defined
PARMS :

RETURN : None

ActivitiesForOneYear 0 :

METHOD : ActivitiesForOneYear
PURPOSE : load up the activities for the calendar and calendar year selected.
PARMS :

RETURN : None

AllYears 0 :

METHOD : AllYears
PURPOSE : load the year dropdown box, default value is the current year
PARMS :

RETURN : None

AddNewYear 0 :

METHOD : AddNewYear
PURPOSE : add a year to the year dropdown used to select the year being viewed / updated
PARMS :

RETURN : None

---

syncDetails () :

=

METHOD : syncDetails
PURPOSE : this sub will make sure the fiscal periods, activities and activity types are all synchronized to the calendar and year selected.
PARMS :

RETURN : None

=

Form_Load () :

=

METHOD : Form_Load
PURPOSE :'load the form'
dim thelocal variables, set the clientside cursor, load the calendar grid
PARMS :

RETURN : None

=

dgrdAllCalendars_RowColChange (LastRow : Variant, LastCol : Integer) :

=

METHOD : dgrdAllCalendars_RowColChange
PURPOSE :'load the fiscal months and Activities for the calendar selected.
PARMS :
LastRow [Variant] =
LastCol [Integer] =
RETURN : None

=

dgrdAllCalendars_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :

=

METHOD : dgrdAllCalendars_MouseDown
PURPOSE :'Bring up the context menu for the Calendar Grid
PARMS :
Button [Integer] =
Shift [Integer] =
X [Single] =
Y [Single] =
RETURN : None

=

dgrdActivities_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :

=

METHOD : dgrdActivities_MouseDown
PURPOSE :'Bring up the context menu for the Calendar Grid
PARMS :
Button [Integer] =
Shift [Integer] =
X [Single] =
Y [Single] =
RETURN : None

LOGICAL VIEW REPORT

cboYear_Click () :

METHOD : cboYear_Click
PURPOSE :'check to see if an update has been done, then sync details
PARMS :

RETURN : None

Form_Unload (Cancel : Integer) :

METHOD : Form_Unload
PURPOSE :'set the globals to nothing
PARMS :
Cancel [Integer] =
RETURN : None

Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :
PARMS :
Cancel [Integer] =
UnloadMode [Integer] =
RETURN : None

dgrdAllCalendars_PostEvent (MsgId : Integer) :

METHOD : dgrdAllCalendars_PostEvent
PURPOSE : This method will refresh the datasets and the grids
PARMS :
MsgId [Integer] =
RETURN : None

dgrdAllCalendars_BeforeRowColChange (Cancel : Integer) :
PARMS :
Cancel [Integer] =
RETURN : None

dgrdActivities_RowColChange (LastRow : Variant, LastCol : Integer) :
PARMS :
LastRow [Variant] =
LastCol [Integer] =
RETURN : None

dgrdActivities_PostEvent (MsgId : Integer) :

METHOD : dgrdActivities_PostEvent
PURPOSE : The Post Event process will refresh the record set for the grid

LOGICAL VIEW REPORT

PARMS :
Msgid [Integer] =
RETURN : None

---

dgrdActivities_BeforeRowColChange (Cancel : Integer) :
PARMS :
Cancel [Integer] =
RETURN : None

---

dcboActivityTypes_Click (Area : Integer) :

METHOD : dcboActivityTypes_Click
PURPOSE : Will move the RS to get current.
PARMS :
Area [Integer] =
RETURN : None

---

dcboActivityDates_Click (Area : Integer) :

METHOD : dcboActivityDates_Click
PURPOSE : Will move the RS to get current.
PARMS :
Area [Integer] =
RETURN : None

---

abarContextMenu_Click (Tool : ActiveBarLibraryCtl.Tool) :

METHOD : abarContextMenu_Click
PURPOSE : This sub will be fired when there is a click on any of the context menu's for this
form. Identify the appropriate click and take the appropriate action.
PARMS :
Tool [ActiveBarLibraryCtl.Tool] =
RETURN : None

[redacted]

MODULE : frmCalendarActivityType
PURPOSE : This form is used to maintain Activity Types that may be associated with calendars.
add, update or delete an activity type name or description

LOGICAL VIEW REPORT

Private Attributes:

cFORM_MIN_HEIGHT : = 6615
cFORM_MIN_WIDTH : = 8835
Some Form Constants

Private Operations:

ValidateForm (aAction : frmActivityTypeRSActions) : Long

=

METHOD : ValidateForm
PURPOSE : validate the data that has changed, if any
PARMS :
aAction [frmActivityTypeRSActions] =
RETURN : Long

=

processRS_ActivityTypes (aAction : frmActivityTypeRSActions, avParms() : Variant) : Long

=

METHOD : processRS_ActivityTypes
PURPOSE : determine if there are any changes to the recordset. If there are any changes
then edit the cahgnes and save the RS.
This also controls the button enable/disable logic
PARMS :
aAction [frmActivityTypeRSActions] =
avParms [Variant] =
RETURN : Long

=

Form_Unload (Cancel : Integer) :

=

METHOD : Form_Unload
PURPOSE : form, form, go away,
use the recordsets another day
PARMS :
Cancel [Integer] =
RETURN : None

=

Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :

=

METHOD : Form_QueryUnload
PURPOSE : preprocessing for form unload.
save the RS if necessary
PARMS :
Cancel [Integer] =
UnloadMode [Integer] =
RETURN : None

=

populate_ReservedActivityTypes 0 :

=

-C13-

LOGICAL VIEW REPORT

METHOD : populate_ReservedActivityTypes
PURPOSE : load the reserved activity type grid
PARMS :

RETURN : None

populate_ActivityTypes 0 :

METHOD : populate_ActivityTypes
PURPOSE : load or refresh the activty type grid
PARMS :

RETURN : None

Form_Load 0 :

METHOD : Form_Load
PURPOSE :
PARMS :

RETURN : None

dgrdActivityType_PostEvent (MsgId : Integer) :

METHOD : dgrdActivityType_PostEvent
PURPOSE : 'Check to see if the work on the previous row needs to be changed
PARMS :
MsgId [Integer] =
RETURN : None

dgrdActivityType_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :

METHOD : dgrdActivityType_MouseDown
PURPOSE : bring up the context menu on right click over grid.
PARMS :
Button [Integer] =
Shift [Integer] =
X [Single] =
Y [Single] =
RETURN : None

dgrdActivityType_BeforeRowColChange (Cancel : Integer) :

METHOD : dgrdActivityType_BeforeRowColChange
PURPOSE : if there has been any change in the grid the call processRs to edit and update the RS
PARMS :

LOGICAL VIEW REPORT

Cancel [Integer] =
RETURN : None

=

abarPopupMenu_Click (Tool : ActiveBarLibraryCtl.Tool) :

=

METHOD : abarPopupMenu_Click
PURPOSE : 'This controls the behaviour of the ActiveBar menu control.
delete: delete the row and select the first row in the recordset.
add: add a new row to the recordset.
PARMS :
Tool [ActiveBarLibraryCtl.Tool] =
RETURN : None

=

frmCurrFiscalPeriod

=

MODULE : frmCurrFiscalPeriod
PURPOSE : this form is used to update the current fiscal period for any calendar
defined in the AE.

=

Private Attributes:

cFORM_MIN_HEIGHT : = 2460
cFORM_MIN_WIDTH : = 9690
Some Form Constants
glCalendarID : Long

Public Operations:

CalendarID (alCalendarId : Long) :

=

METHOD : CalendarID
PURPOSE : This is the setter method for the CalendarId
PARMS :
alCalendarId [Long] =
RETURN : None

=

CalendarID () : Long

=

METHOD : CalendarID
PURPOSE : This is a getter method for the CalendarId Property
PARMS :

RETURN : Long

=

LOGICAL VIEW REPORT

Private Operations:

Form_Unload (Cancel : Integer) :

=

METHOD : Form_Unload
PURPOSE : save any changes.
There are no edits that need to be performed. The user can only change a month or year,
using bound dropdowns that will not allow an error (thats my story, and I'm sticking to it.)
PARMS :
Cancel [Integer] =
RETURN : None

=

SaveCalendars 0 : Variant

=

METHOD : SaveCalendars
PURPOSE : if the fiscal period for any calendar has been changed, then save the changes
PARMS :

RETURN : Variant

=

PopulateCalendars 0 :

=

METHOD : PopulateCalendars
PURPOSE : get all the calendar and put them in the calendar grid for display.
PARMS :

RETURN : None

=

Form_Load 0 :

=

METHOD : Form_Load
PURPOSE : load the form, get the calendars
PARMS :

RETURN : None

=

frmBWenModMgmt

Private Attributes:

cFORM_MIN_HEIGHT : = 3540
cFORM_MIN_WIDTH : = 10815

C15

LOGICAL VIEW REPORT

Public Operations:

cancelAdd () :
deleteData () :
newData () :

Private Operations:

ClearSearchFields () :
ManageSearchButton () :
EnableEditFields (bEnable : Boolean) :
ProcessRS_EventMods (aAction : frmEventModActions, avParms() : Variant) : frmEventModRC
cmdSearch_Click () :
****
****
cmdClear_Click () :
txtModifierName_Change () :
txtModifierDescrip_Change () :
Form_Unload (Cancel : Integer) :
Form_Load () :
dgrdEventMods_RowColChange (LastRow : Variant, LastCol : Integer) :
dgrdEventMods_PostEvent (MsgId : Integer) :
dgrdEventMods_MouseUp (Button : Integer, Shift : Integer, X : Single, Y : Single) :
dgrdEventMods_BeforeRowColChange (Cancel : Integer) :
sbarPopMenus_Click (Tool : ActiveBarLibraryCtl.Tool) :

frmFiscalPeriodStartDates

Private Attributes:

cActivityTypeFiscalStart : = 1
cFORM_MIN_HEIGHT : = 3500
cFORM_MIN_WIDTH : = 8500
Some Form Constants
giCalendarID : Long
gstrCalendarName : String
Dim globals, private to form

Public Operations:

CalendarID (aiCalendarId : Long) :
CalendarName (astrCalendarName : String) :

Private Operations:

ValidateForm (aAction : frmCalendarRSActions) : Long
==========
=
METHOD : ValidateForm
PURPOSE : check each recordset that can be updated to see if it has changed.
if it has changed, then validate the data entered
if validations are passed, then call the appropriate save routine
PARMS :
aAction [frmCalendarRSActions] =
RETURN : Long
==========
=

LOGICAL VIEW REPORT

```
processRS_Calendar (aAction : frmCalendarRSActions, avParms() : Variant) : Long
=
METHOD : processRS_Calendar
PURPOSE : This is the brains of the operation. When it is called it:
      1. checks the action used to call it
      2. then it enables and disbles controls
      3. checks recordsets to see if they need to be validated and saved
PARMS :
      aAction [frmCalendarRSActions] =
      avParms [Variant] =
RETURN : Long
=
SaveYear () : Variant
=
METHOD : SaveYear
PURPOSE : 'This function will save any changes made to the starting periods for the
Fiscal Year.
PARMS :

RETURN : Variant
=
UnLockFiscalYear () :
=
METHOD : UnLockFiscalYear
PURPOSE :
PARMS :

RETURN : None
=
LockFiscalYear () :
=
METHOD : LockFiscalYear
PURPOSE : lock and disable the fiscal year until a calendar year is selected.
PARMS :

RETURN : None
=
AllFiscalMonthsforCalendar () :
=
METHOD : AllFiscalMonthsforCalendar
PURPOSE : load up the twelve fiscal periods for the calendar and calendar year selected
PARMS :

RETURN : None
=
```

LOGICAL VIEW REPORT

```
AllYears () :
================================================================================
=
METHOD : AllYears
PURPOSE :'load the year dropdown box, default value is the current year
PARMS :

RETURN : None
================================================================================
=
AddNewYear () :
================================================================================
=
METHOD : AddNewYear
PURPOSE : add a year to the year dropdown used to select the year being viewed /
updated
PARMS :

RETURN : None
================================================================================
=
Form_Load () :
================================================================================
=
METHOD : Form_Load
PURPOSE :'load the form'
        dim thelocal variables, set the clientside cursor,
PARMS :

RETURN : None
================================================================================
=
cboYear_Click () :
================================================================================
=
METHOD : cboYear_Click
PURPOSE :'check to see if an update has been done, then sync details
PARMS :

RETURN : None
================================================================================
=
pvdFiscalMonth_Validate (Index : Integer, Cancel : Boolean) :
================================================================================
=
METHOD : pvdFiscalMonth_Validate
PURPOSE :'if the fiscal period is changing move the data to the RS for update.
 the RS is indexed relative to 1, the pvdFiscalMonth array is indexed relative to 0
PARMS :
        Index [Integer] =
        Cancel [Boolean] =
RETURN : None
================================================================================
=
Form_Unload (Cancel : Integer) :
================================================================================
=
```

LOGICAL VIEW REPORT

METHOD : Form_Unload
PURPOSE :' set the globals to nothing
PARMS :
Cancel [Integer] =
RETURN : None

=

Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :
PARMS :
Cancel [Integer] =
UnloadMode [Integer] =
RETURN : None

=

fraFiscalYear_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :

=

METHOD : fraFiscalYear_MouseDown
PURPOSE : Determine if the left button was clicked then set the context menu accordingly
PARMS :
Button [Integer] =
Shift [Integer] =
X [Single] =
Y [Single] =
RETURN : None

=

Form_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :

=

METHOD : Form_MouseDown
PURPOSE : Determine if the left button was clicked then set the context menu accordingly
PARMS :
Button [Integer] =
Shift [Integer] =
X [Single] =
Y [Single] =
RETURN : None

=

pvdFiscalMonth_MouseDown (Index : Integer, Button : Integer, Shift : Integer, X : Single, Y : Single) :

=

METHOD : pvdFiscalMonth_MouseDown
PURPOSE : Determine if the left button was clicked then set the context menu accordingly
PARMS :
Index [Integer] =
Button [Integer] =
Shift [Integer] =
X [Single] =
Y [Single] =
RETURN : None

=

LOGICAL VIEW REPORT

abarContextMenu_Click (Tool : ActiveBarLibraryCtl.Tool) :

=

METHOD : abarContextMenu_Click
PURPOSE : This sub will be fired when there is a click on any of the context menu's for this
form. Identify the appropriate click and take the appropriate action.
PARMS :
Tool [ActiveBarLibraryCtl.Tool] =
RETURN : None

=

frmJEMaintenance

Private Attributes:

cFORM_MIN_HEIGHT : = 7140
cFORM_MIN_WIDTH : = 10695
Some Form Constants
gbJEChange : Boolean
glJeID : Long

Public Operations:

addNewDRCRPair 0 :
deleteData 0 :
newData 0 :
cancelDrCr 0 :
cancelJe 0 :

=

METHOD : cancelJe
PURPOSE : This method will cancel a new record for JE Header pairs
after the user clicked cancel from the context menu. It will
set the context menu and enable or disable all fields
PARMS :
RETURN :

=

save 0 :

```
**************************************************************************************
************************
Private Sub txtJENumber_KeyUp(KeyCode As Integer, Shift As Integer)
'PURPOSE : Check to insure that some search criteria have been entered.
'       If not, disable the search button
 If Len(Trim(txtJEName.Text)) < 3 And Trim(txtJENumber.Text) = "" Then
   cmdSearch.Enabled = False
 Else
   cmdSearch.Enabled = True
 End If
End Sub
**************************************************************************************
************************
```

LOGICAL VIEW REPORT

Private Operations:

enableDRCRGrid (bEnable : Boolean) :
ClearSearchFields () :
ManageSearchButton () :
EnableEditFields (bEnable : Boolean) :
txtUJeNumber_KeyPress (KeyAscii : Integer) :
txtUJEName_KeyPress (KeyAscii : Integer) :
txtUDescrip_KeyPress (KeyAscii : Integer) :
Reset_Errors () :

populate_JE_Headers (aInitGrid : Boolean = False) :

processRS_SLCOA (aAction : frmJeRSActions, avParms() : Variant) : Long

processRS_DRCRPairs (aAction : frmJeRSActions, avParms() : Variant) : Long
processRS_JEHeaders (aAction : frmJeRSActions, avParms() : Variant) : Long

METHOD : processRS_JEHeaders
PURPOSE : Any Movement(or add/delete) in the J/E Header Recordset
is processed here. Before we process the change, we check
to see if we need to save the information associated with
the currently active row. Before saving any information,
the content of the columns is validated. This method will
also make calls to child recordset processes of the same
type.
PARMS :
aAction [frmJeRSActions] = see Enum in GenDecs for details
avParms [Variant] = if any parms need to be passed in
RETURN : Long (0=success, -1=failure)

txtJENumber_KeyPress (KeyAscii : Integer) :
txtJENumber_Change () :
txtJEName_Change () :
tdbcDebitSL_ItemChange () :
tdbcCreditSL_ItemChange () :
Form_Unload (Cancel : Integer) :
dgrdJournalEntries_RowColChange (LastRow : Variant, LastCol : Integer) :

dgrdJournalEntries_BeforeRowColChange (Cancel : Integer) :

cmdSearch_Click () :

cmdClear_Click () :

sbarContentMenu_Click (Tool : ActiveBarLibraryCtl.Tool) :

LOGICAL VIEW REPORT

Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :
****
****
Form_Load () :
****
****
dgrdJournalEntries_PostEvent (MsgId : Integer) :
dgrdJournalEntries_MouseUp (Button : Integer, Shift : Integer, X : Single, Y : Single) :
dgrdJEDRCRPairs_RowColChange (LastRow : Variant, LastCol : Integer) :
dgrdJEDRCRPairs_MouseUp (Button : Integer, Shift : Integer, X : Single, Y : Single) :
dgrdJEDRCRPairs_BeforeRowColChange (Cancel : Integer) :
****
****

frmOfficeMaint

Private Attributes:

cFORM_MIN_HEIGHT : = 3630
cFORM_MIN_WIDTH : = 9600
gbUpdateOffice : Boolean
gbAddNewOffice : Boolean

Public Operations:

newData () :
deleteData () :

Private Operations:

cancelOffice () :

=

METHOD : cancelOffice
PURPOSE : This method will cancel an update to the record set
PARMS :

RETURN : None

=

newOffice () :

PARMS :

RETURN : None

=

disableText () :

=

METHOD : disableText
PURPOSE : This will disable all the fields
PARMS :

RETURN : None

=

LOGICAL VIEW REPORT

enableText 0 :

=

METHOD : enableText
PURPOSE : This will enable all the fields
PARMS :

RETURN :

=

sndPanelMsg (amsgType : panelMsg) :

=

METHOD : sndPanelMsg
PURPOSE : This will display the appropriate message to the panel
PARMS :
amsgType [panelMsg] = the type of message that should be displayed
RETURN : None

=

bldBusinesTrans 0 :

=

METHOD : bldBusinesTrans
PURPOSE : This will build the translation tables within the grid
PARMS :

RETURN : None

=

getAllBusinessData 0 :

=

METHOD : getAllBusinessData
PURPOSE : this method will get all the businesses from the data base
PARMS :

RETURN : None

=

Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :
deleteOffice 0 :

=

METHOD : deleteOffice
PURPOSE : This will delete a record from the data base
PARMS :

RETURN :

=

saveOffice 0 :

=

METHOD : saveOffice
PURPOSE : This will determine whether the record will be inserted or updated.
A request will be made to the business services

LOGICAL VIEW REPORT

PARMS :

RETURN :

---

checkForMissingData () : Boolean

METHOD : checkForcheckForMissingData
PURPOSE : This will determine if any required data is missing.
PARMS :

RETURN : lbBad As Boolean

---

setTextFields () :

METHOD : setTextFields
PURPOSE : This will bind the data
PARMS :

RETURN :

Form_Load () :

METHOD : Form_Load
PURPOSE : This will retrieve all reqired data, then bind
PARMS :

RETURN :

---

dgrdOffice_RowColChange (LastRow : Variant, LastCol : Integer) :

METHOD : dgrdOffice_RowColChange
PURPOSE : This will post the event based on whether a record was updated
PARMS :

RETURN :

---

dgrdOffice_PostEvent (MsgId : Integer) :

METHOD : dgrdOffice_PostEvent
PURPOSE : This message will trigged from the RowColChange. If data was changed
the event will be posted. It is here will the grid will be re-built.
PARMS :

RETURN :

LOGICAL VIEW REPORT

dgrdOffice_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :

=

METHOD : dgrdOffice_MouseDown
PURPOSE : This will determine if the right mouse was pressed, if so the appropriate active bar will be displayed
PARMS :

RETURN :

=

dgrdOffice_BeforeRowColChange (Cancel : Integer) :

=

METHOD : dgrdOffice_BeforeRowColChange
PURPOSE : This event will determine if data is missing. If ok the data will saved
PARMS :

RETURN :

=

dcboBusiness_Change () :
PARMS :

RETURN : None

=

abarOffice_Click (Tool : ActiveBarLibraryCtl.Tool) :
PARMS :
Tool [ActiveBarLibraryCtl.Tool] =
RETURN : None

=

getAllOfficeData () :

=

METHOD : getAllOfficeData
PURPOSE : This will get all the records in the office table
PARMS :

RETURN :

=

frmOfficeMainPage

Private Attributes:

cFORM_MIN_HEIGHT : = 6150
cFORM_MIN_WIDTH : = 10050
gbUpdateOffice : Boolean
gbAddNewOffice : Boolean

LOGICAL VIEW REPORT

Public Operations:

deleteOffice () :
saveOffice () :
missingData () : Boolean
setTextFields () :
getAllOfficeData () :

Private Operations:

Form_Unload (Cancel : Integer) :
Form_Load () :
dgrdOffice_RowColChange (LastRow : Variant, LastCol : Integer) :
dgrdOffice_PostEvent (MsgId : Integer) :
dgrdOffice_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :
dgrdOffice_BeforeRowColChange (Cancel : Integer) :
sbarOffice_Click (Tool : ActiveBarLibraryCtl.Tool) :

frmOrganizationMaint

Private Attributes:

cFORM_MIN_HEIGHT : = 5325
cFORM_MIN_WIDTH : = 10050
gbUpdateCorp : Boolean
gbAddNewOffice : Boolean
gbAddNewCorp : Boolean
gstrSearchCriteria : String
gstrRowName : String

Public Operations:

deleteData () :

=

METHOD : deleteData
PURPOSE : Called from the MDI form, this will call this forms delete method
PARMS :

RETURN :

=

newData () :

=

METHOD : newData
PURPOSE : Called from the MDi form this will determine if data can be saved
prior to createing a new record
PARMS :

RETURN :

=

LOGICAL VIEW REPORT

Private Operations:

checkForMissingOffice 0 : Boolean

getAllOfficeData 0 :

METHOD : getAllOfficeData
PURPOSE : This will get all the office data
PARMS :

RETURN :

deleteOfficeCorp 0 :

METHOD : deleteOfficeCorp
PURPOSE : This will delete the Office Corp
PARMS :

RETURN :

enableText 0 :

METHOD : enableText
PURPOSE : This will enable all the fields
PARMS :

RETURN :

disableText 0 :

cancelOfficeForCorpOrg 0 :

METHOD : cancelOfficeForCorpOrg
PURPOSE : This will cancel an add from the record set
PARMS :

RETURN :

newOfficeForCorpOrg 0 :

METHOD : newOfficeForCorpOrg
PURPOSE : This will add a new OfficeCorp to the Record set
PARMS :

RETURN :

cancelCorpOrg 0 :

METHOD : cancelCorpOrg

LOGICAL VIEW REPORT

PURPOSE : This will cancel an add to the corporg Record set
PARMS :

RETURN :

---

deleteCorpOrg 0 :

METHOD : deleteCorpOrg
PURPOSE : This will delete a record from the Corp_Org Table
PARMS :

RETURN :

---

bldCalendarTranslation 0 :

METHOD : bldCalendarTranslation
PURPOSE : This will build the calendar translation table
PARMS :

RETURN :

Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :
saveOfficeCorpRS 0 :
PARMS : None

RETURN : None

---

saveOfficeCorp 0 :

METHOD : saveOfficeCorp
PURPOSE : This will determine if the data should inserted or updated.
The appropriate call to the business service will be made
PARMS :

RETURN :

---

checkForMissingData 0 : Boolean

METHOD : checkForMissingData
PURPOSE : This method will determine if any required fields are missing
PARMS :

RETURN : lbBad As Boolean

LOGICAL VIEW REPORT

newCorpOrg () :
sndPanelMsg (amsgType : panelMsg) :

=

METHOD : sndPanelMsg
PURPOSE : This will display the appropriate message to the panel
PARMS :
amsgType [panelMsg] = the type of message that should be displayed
RETURN : None

=

getAllCorpOrg () :

=

METHOD : getAllCorpOrg
PURPOSE : This will get all the Corporations
PARMS :

RETURN :

=

getOfficesForCorp (aSearch : groupID) : Variant

=

METHOD : getOfficesForCorp
PURPOSE : This will get all the officeGroup data
PARMS :

RETURN :

=

bldOfficeTranslation () :
setTextFields () :

=

METHOD : setTextFields
PURPOSE : This method will bind the text fields and drop downs to the data
PARMS :

RETURN :

=

Form_Load () :

=

METHOD : Form_Load
PURPOSE : This event will retrieve the appropriate data, build the grid
set the drop downs and build translations
PARMS :

RETURN :

=

dgrdOffices_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :
dgrdOffices_BeforeRowColChange (Cancel : Integer) :
dgrdCorpOrg_RowColChange (LastRow : Variant, LastCol : Integer) :

=

LOGICAL VIEW REPORT

METHOD : dgrdCorpOrg_RowColChange
PURPOSE : If data changed the method will post an event
PARMS :

RETURN :

---

dgrdCorpOrg_PostEvent (MsgId : Integer) :

METHOD : dgrdCorpOrg_PostEvent
PURPOSE : This event is posted in the RowColChange event. If data was changed this event will re-populate the grid.
PARMS :

RETURN :

---

dgrdCorpOrg_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :

METHOD : dgrdCorpOrg_MouseDown
PURPOSE :Determine if the right mouse button was selected. If so will then set the options appropriately.
PARMS :

RETURN :

---

dgrdCorpOrg_BeforeRowColChange (Cancel : Integer) :

METHOD : dgrdCorpOrg_BeforeRowColChange
PURPOSE : Determine if any data missing if none the data will be saved. The appropriate message will be sent if any data missing.
PARMS :

RETURN :

---

dcboOffice_Click (Area : Integer) :

METHOD : dcboOffice_Click
PURPOSE : This will update the Recordset that this drop down is attached to
PARMS :

RETURN :

---

dcboCalendar_Click (Area : Integer) :

METHOD : dcboCalendar_Click
PURPOSE : This will update the Recordset that this drop down is attached to
PARMS :

LOGICAL VIEW REPORT

RETURN :

=

abarOrganization_Click (Tool : ActiveBarLibraryCtl.Tool) :

=

METHOD : abarOrganization_Click
PURPOSE : This will determine what option was selected on the Action Bar
PARMS :

RETURN :

=

frmParmMaint

Private Attributes:

cFORM_MIN_HEIGHT : = 3720
cFORM_MIN_WIDTH : = 9210
gbUpdateParm : Boolean
gbAddNewParm : Boolean

Private Operations:

Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :
saveParm () :

=

METHOD : saveOfficeCorp
PURPOSE : This will determine if the data should inserted or updated.
The appropriate call to the business service will be made
PARMS :

RETURN :

=

cancelParm () :

=

METHOD : cancelCorpParm
PURPOSE : This will cancel an add to the corpParm Record set
PARMS :

RETURN :

=

deleteParm () :

=

METHOD : deleteCorpParm
PURPOSE : This will delete a record from the Corp_Parm Table
PARMS :

RETURN :

LOGICAL VIEW REPORT

=

newParm () :

sndPanelMsg (amsgType : panelMsg) :

=

METHOD : sndPanelMsg
PURPOSE : This will display the appropriate message to the panel
PARMS :
amsgType [panelMsg] = the type of message that should be displayed
RETURN : None

=

checkForMissingData () : Boolean

=

METHOD : checkForMissingData
PURPOSE : This method will determine if any required fields are missing
PARMS :

RETURN : lbBad As Boolean

=

setTextFields () :

=

METHOD : setTextFields
PURPOSE : This method will bind the text fields and drop downs to the data
PARMS :

RETURN :

=

disableText () :
enableText () :

=

METHOD : enableText
PURPOSE : This will enable all the fields
PARMS :

RETURN :

=

bldParmTranslation () :

=

METHOD : bldParmTranslation
PURPOSE : This will build the type translation table
PARMS :

RETURN :

=

getAllParms () :

=

METHOD : getAllCorpParm
PURPOSE : This will get all the Corporations
PARMS :

RETURN :

=

Form_Load () :
dgrdParm_RowColChange (LastRow : Variant, LastCol : Integer) :
dgrdParm_PostEvent (MsgId : Integer) :
dgrdParm_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :
dgrdParm_BeforeRowColChange (Cancel : Integer) :
sbarParm_Click (Tool : ActiveBarLibraryCtl.Tool) :

frmProductPick

Public Attributes:

mvarBusinessEventID : Long
mvarProductID : Long
mvarProductName : String

Public Operations:

BusinessEventID (aBusEventID : Long) :
BusinessEventID () : Long
ProductId () : Long
ProductName () : String

Private Operations:

Form_Activate () :
distProducts_MouseDown (Button : Integer, Shift : Integer, X : Single, Y : Single) :
distProducts_KeyPress (KeyAscii : Integer) :
distProducts_Click () :

frmQualEventInquiry

Private Attributes:

gErrDesc : String
gErrSource : String
gErrNum : Long
gDBErrorFlg : Boolean

Other Global Flotsam

cIDSTR : String = "ID:"
cFORM_MIN_HEIGHT : = 7530
cFORM_MIN_WIDTH : = 11880

Constants

LOGICAL VIEW REPORT

Private Operations:

Display_Error_Msg () :
refresh_subordinate_grids (alAction : Integer) :
Sync_TreeToRS () :
pvtvEventProd_LButtonUp (node : PVTreeView3Lib.PVIBranch, X : Single, Y : Single) :
pvtvEventProd_AfterSelChange (node : PVTreeView3Lib.PVIBranch) :
dgrdQELines_RowColChange (LastRow : Variant, LastCol : Integer) :
sbarPopMenus_Click (Tool : ActiveBarLibraryCtl.Tool) :
Form_Load () :

frmQualEventLines

Private Attributes:

cINVALIDCALLTOEVENTLINES : Variant = cHIGHESTERROR + 1000
cHIGHESTERROR : Variant = vbObjectError + 256
cFORM_MIN_HEIGHT : = 6345
cFORM_MIN_WIDTH : = 10485
mvarProductID : Long
mvarBusEventID : Long

Form Global variables and constants

Public Operations:

enableText () :
disableText () :
setQEPopup () :
cancelQEAdd () :
setEventLinePopup () :
cancelEventLineAdd () :
ProductId (alProdID : Long) :
ProductId () : Long
BusinessEventID (alBusEvID : Long) :
BusinessEventID () : Long

Private Operations:

Form_Unload (Cancel : Integer) :
Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :
Reset_Errors () :
ComboBox_Change_Common () :
ProcessRS_QEBooksets (aAction : frmQELinesActions, avParms() : Variant) : frmQELinesResponses
ProcessRS_QELines (aAction : frmQELinesActions, avParms() : Variant) : frmQELinesResponses
Form_Load () :
dgrdQELines_MouseUp (Button : Integer, Shift : Integer, X : Single, Y : Single) :
dgrdQELines_PostEvent (MsgId : Integer) :
dgrdQELines_BeforeRowColChange (Cancel : Integer) :
dgrdBooksets_RowColChange (LastRow : Variant, LastCol : Integer) :
dgrdBooksets_MouseUp (Button : Integer, Shift : Integer, X : Single, Y : Single) :
dcboRule_Change () :
dcboNonEarningJE_Change () :
dcboEventMod_Change () :
dcboEarningJE_Change () :
sbarPopMenus_Click (Tool : ActiveBarLibraryCtl.Tool) :

frmQualEventMaint

Private Attributes:

cFORM_MIN_HEIGHT : = 3750
cFORM_MIN_WIDTH : = 10845

Public Operations:

cancelAdd () :
deleteData () :
newData () :

Private Operations:

Show_Product_Window () :
txtEventName_Validate (Cancel : Boolean) :
txtDescrip_Validate (Cancel : Boolean) :
Sync_TreeToRS () :
Reset_Errors () :
processRS_PBE (aAction : frmQERSActions, avParms() : Variant) : Long
pvtvEventProd_RButtonUp (node : PVTreeView3Lib.PVIBranch, X : Single, Y : Single) :
pvtvEventProd_RButtonDown (node : PVTreeView3Lib.PVIBranch, X : Single, Y : Single) :
pvtvEventProd_BeforeSelChange (node : PVTreeView3Lib.PVIBranch, bProcessed : Boolean) :
pvtvEventProd_AfterSelChange (node : PVTreeView3Lib.PVIBranch) :
Form_Unload (Cancel : Integer) :
Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :
sbarPopMenus_Click (Tool : ActiveBarLibraryCtl.Tool) :
Form_Load () :

frmQualEventParmsMaint

Private Attributes:

cFORM_MIN_HEIGHT : = 6150
cFORM_MIN_WIDTH : = 10050
miEventID : Long
miProdId : Long
gbUpdateParm : Boolean
gbAddNewParm : Boolean

Public Operations:

getAllParms () :
bldParmTranslation () :
eventID (aleventID : Variant) :
eventID () : Variant
ProductId (aiProdID : Variant) :
ProductId () : Variant
getAllQEParms (aiProdID : Long, aleventID : Long) :

=
METHOD : getAllQEParms
PURPOSE : This will get all the Qualified Event Parms
PARMS :

RETURN :
=

LOGICAL VIEW REPORT

Private Operations:

Form_Load () :
dgrdQEParms_RowColChange (LastRow : Variant, LastCol : Integer) :

frmRuleMaint

Private Attributes:

cFORM_MIN_HEIGHT : = 3540
cFORM_MIN_WIDTH : = 10575

Public Operations:

cancelAdd () :
deleteData () :
newData () :

Private Operations:

GetSearchFields () :
ClearSearchFields () :
ManageSearchButton () :
EnableEditFields (bEnable : Boolean) :
processRS_Rules (nAction : frmRuleMaintActions, avParms() : Variant) : frmRuleMaintRC
cmdSearch_Click () :
****
****
cmdClear_Click () :
txtRuleName_Change () :
txtRuleDescrip_Change () :
Form_Unload (Cancel : Integer) :
Form_Load () :
dgrdRules_RowColChange (LastRow : Variant, LastCol : Integer) :
dgrdRules_PostEvent (MsgId : Integer) :
dgrdRules_MouseUp (Button : Integer, Shift : Integer, X : Single, Y : Single) :
dgrdRules_BeforeRowColChange (Cancel : Integer) :
sbarPopMenus_Click (Tool : ActiveBarLibraryCtl.Tool) :

frmRuleMaint (RS)

Private Attributes:

cINVALIDCALLTORULELINES : Variant = cHIGHESTERROR + 1000
cHIGHESTERROR : Variant = vbObjectError + 256
cFORM_MIN_HEIGHT : = 5865
cFORM_MIN_WIDTH : = 9735
mvarRuleAEID : Long
Form Global variables and constants
giNextSeq : Integer

-C37-

LOGICAL VIEW REPORT

Public Operations:

Disable_Edit_Fields () :
Reset_Edit_Fields () :
updateContextMenu () :
Rule_ID (aIID : Long) :
Rule_ID () : Long

Private Operations:

add_listbox_item (aiVarType : frmRuleVarTypes, avItem : Variant, aiPosition : Variant) :
resequence_existing_lines (aNextSeq : Integer, astrSeqType : String) :
build_current_line () :
processRS_RuleVars (aAction : frmRuleRSActions, avParms() : Variant) : frmRuleProcessRC
processRS_RuleLines (aAction : frmRuleRSActions, avParms() : Variant) : frmRuleProcessRC
cmdSearch_Click () :
dgrdRuleLines_PostEvent (MsgId : Integer) :
dgrdRuleLines_BeforeRowColChange (Cancel : Integer) :
dgrdRuleLines_MouseUp (Button : Integer, Shift : Integer, X : Single, Y : Single) :
sbarPopMenus_Click (Tool : ActiveBarLibraryCtl.Tool) :
dcboDestination_Click (Area : Integer) :
dcboRuleAction_Click (Area : Integer) :
dcboPriorSequence_Click (Area : Integer) :
dcboInputParm_Click (Area : Integer) :
txtConstantValue_Validate (Cancel : Boolean) :
dcboDBField_Click (Area : Integer) :
lstRuleVars_Click () :
Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :
Form_Load () :
Form_Activate () :
cmdNewParm_Click () :

frmSLBalancesRpt

Private Attributes:

cPRINTFTR : String = "Confidential and Proprietary Information - GE Capital Commercial Equipment Financing"
cPRINTHDR : String = "GE Capital CEF - Accounting Engine"

Constants for Print Header and Footer

cFORM_MIN_HEIGHT : = 7815
cFORM_MIN_WIDTH : = 10770

Constants

Private Operations:

SetPrintInfo_Details () :
SetPrintInfo_Balances () :
Sync_Detail_Window (aiBalanceID : Long) :
dgrdSLDetail_MouseUp (Button : Integer, Shift : Integer, X : Single, Y : Single) :
dgrdSLBals_RowColChange (LastRow : Variant, LastCol : Integer) :
dgrdSLBals_MouseUp (Button : Integer, Shift : Integer, X : Single, Y : Single) :

=

Balance and Detail Grid Code

=

sbarContextMenu_Click (Tool : ActiveBarLibraryCtl.Tool) :

=

Active Bar Code

LOGICAL VIEW REPORT

cmdFetch_Click 0 :
cmdClear_Click 0 :

Command Button Code

Form_Load 0 :

Form_Load

frmSLChartofAccounts

MODULE : frmSLChartofAccounts
PURPOSE : This form will add, update and delete subledger chart of accounts

Private Attributes:

cFORM_MIN_HEIGHT : = 7020
cFORM_MIN_WIDTH : = 10650
gstrSearchCriteria : String
gstrRowName : String

Public Operations:

deleteData 0 :

METHOD : deleteData
PURPOSE : This method is called by the MDI form from the tool bar
PARMS :

RETURN : None

newData 0 :
PARMS :

RETURN : None

Private Operations:

disableText 0 :

METHOD : disableText

LOGICAL VIEW REPORT

PURPOSE : This will disable all controls
PARMS :

RETURN : None

---

cancelUpdate () :

METHOD : cancelUpdate
PURPOSE : This will cancel an update
PARMS :

RETURN : None

---

anyChange () : Boolean
PARMS :

RETURN : Boolean

---

checkAcctRoll () :

METHOD : checkAcctRoll
PURPOSE : Need to check to see if the account rollup grid changed
PARMS :

RETURN : None

---

checkTransfer () :

METHOD : checkTransfer
PURPOSE : This method will check to see if the check transfer grid changed
PARMS :

RETURN : None

RecordsetChanges (arsToBeChecked : ADOR.Recordset) : Boolean

METHOD : RecordsetChanges
PURPOSE : This method will check to see if the recordset has changed
PARMS :
arsToBeChecked [ADOR.Recordset] =
RETURN : Boolean

---

bldAfterSave () :

METHOD : bldAfterSave
PURPOSE : This will re-build the grid after saving the data

LOGICAL VIEW REPORT

PARMS :

RETURN : None

save 0 :

METHOD : save
PURPOSE : This method will call the method that will save the Record Set
PARMS :

RETURN : None

getRequestedData 0 :

METHOD : getRequestedData
PURPOSE : This method will get the data the user requested
PARMS :

RETURN : None

saveRs (aCaller : saveActions) : Boolean

METHOD : saveRs
PURPOSE : This method will save the record set information
PARMS :
aCaller [saveActions] =
RETURN : Boolean

bldAcctRoll 0 :

METHOD : bldAcctRoll
PURPOSE : This will build the translation tables for account rollup
PARMS :

RETURN : None

bldTransferTranslate 0 :

METHOD : bldTransferTranslate
PURPOSE : This will build the translation tables for the transfer account
PARMS :

RETURN : None

LOGICAL VIEW REPORT

bldTransferAndRoll 0 :

=

METHOD : bldTransferAndRoll
PURPOSE : This will get all the transfer and roll accounts then build the grid
PARMS :

RETURN : None

=

bldTdgItemList 0 :

=

METHOD : bldTdgItemList
PURPOSE : This will build the Item List
PARMS :

RETURN : None

=

deleteSubLedger 0 :

=

METHOD : deleteSubLedger
PURPOSE : This will delete a subledger
PARMS :

RETURN : None

=

enableText 0 :

=

METHOD : enableText
PURPOSE : This will enable all the controls
PARMS :

RETURN : None

=

checkForMissingData 0 : Variant

=

METHOD : checkForMissingData
PURPOSE : This will determine if there is any missing data
PARMS :

RETURN : Variant

=

newSubledger 0 :

=

METHOD : newSubledger
PURPOSE : This will create a new subledger
PARMS :

LOGICAL VIEW REPORT

RETURN : None

txtSubledgerName_KeyUp (KeyCode : Integer, Shift : Integer) :

METHOD : txtSubledgerName_KeyUp
PURPOSE : This will check to see the values that were entered in this control
PARMS :
KeyCode [Integer] =
Shift [Integer] =
RETURN : None

txtSubLedgerCode_KeyUp (KeyCode : Integer, Shift : Integer) :

METHOD : txtSubLedgerCode_KeyUp
PURPOSE :
PARMS :
KeyCode [Integer] =
Shift [Integer] =
RETURN : None

tdgItemList_RowColChange (LastRow : Variant, LastCol : Integer) :

METHOD : tdgItemList_RowColChange
PURPOSE :
PARMS :
LastRow [Variant] =
LastCol [Integer] =
RETURN : None

tdgItemList_PostEvent (MsgId : Integer) :

METHOD : tdgItemList_PostEvent
PURPOSE :
PARMS :
MsgId [Integer] =
RETURN : None

tdgItemList_MouseUp (Button : Integer, Shift : Integer, X : Single, Y : Single) :

Method : tdgItemList_MouseUp
Purpose: displays the popup, if there are edit checks will not save pointer
Parms : None
Return : None
if there are edit checks what do we want to do

```
tdgItemList_BeforeRowColChange (Cancel : Integer) :
================================================================================
=
METHOD : tdgItemList_BeforeRowColChange
PURPOSE :
PARMS :
        Cancel [Integer] =
RETURN : None
================================================================================
=
tdbgTransfer_Click () :
================================================================================
=
METHOD : tdbgTransfer_Click
PURPOSE : This will re-position the grid if the value is empty
PARMS :

RETURN : None
================================================================================
=
tdbgTransfer_AfterColEdit (ColIndex : Integer) :
================================================================================
=
METHOD : tdbgTransfer_AfterColEdit
PURPOSE :
PARMS :
        ColIndex [Integer] =
RETURN : None
================================================================================
=
tdbgAcctRollUp_Click () :
PARMS :

RETURN : None
================================================================================
=
tdbgAcctRollUp_AfterColEdit (ColIndex : Integer) :
================================================================================
=
METHOD : tdbgAcctRollUp_AfterColEdit
PURPOSE :
PARMS :
        ColIndex [Integer] =
RETURN : None
================================================================================
=
Form_Unload (Cancel : Integer) :
================================================================================
=
METHOD : Form_Unload
PURPOSE :
PARMS :
        Cancel [Integer] =
RETURN : None
================================================================================
=
```

LOGICAL VIEW REPORT

```
Form_Load 0 :
================================================================================
=
METHOD : Form_Load
PURPOSE :
PARMS :

RETURN : None
================================================================================
=
dcboGLMemo_Validate (Cancel : Boolean) :
================================================================================
=
METHOD : dcboGLMemo_Validate
PURPOSE :
PARMS :
      Cancel [Boolean] =
RETURN : None
================================================================================
=
dcboALER_Validate (Cancel : Boolean) :
================================================================================
=
METHOD : dcboALER_Validate
PURPOSE :
PARMS :
      Cancel [Boolean] =
RETURN : None
================================================================================
=
cmdSearch_Click 0 :
================================================================================
==========
Method : cmdSearch_Click
Purpose: This method will build the search criteria
      After returning from the gathering the data the Grid
      will be built
Parms : None
Return : None
================================================================================
==========
cmdClear_Click 0 :
================================================================================
=
METHOD : cmdClear_Click
PURPOSE :
PARMS :

RETURN : None
================================================================================
=
tbarSubledger_Click (Tool : ActiveBarLibraryCtl.Tool) :
```

LOGICAL VIEW REPORT

frmSlChartGroups

=

MODULE : frmSlChartGroups
PURPOSE : This form will allow you to add,update,delete and display subledger groups. Subledgergroups contain subledger chart of accounts

=

Private Attributes:

cFORM_MIN_HEIGHT : = 8160
cFORM_MIN_WIDTH : = 10620
gbUpdateGroup : Boolean
gbAddNewSLChart : Boolean
gbAddNewGroup : Boolean
gstrSearchCriteria : String
gstrRowName : String

Public Operations:

chkForMissingSLChart 0 : Boolean
deleteData 0 :

=

METHOD : deleteData
PURPOSE : The purpose of this method is to delete data
PARMS :

RETURN : None

=

newData 0 :

=

METHOD : newData
PURPOSE : This is called by the MDI tool bar
PARMS :

RETURN : None

=

Private Operations:

disableText 0 :

=

METHOD : disableText
PURPOSE : This method will disable the controls on the window
PARMS :

RETURN : None

=

newSLChartForGroup 0 :
PARMS : None

LOGICAL VIEW REPORT

RETURN : None

reEnable 0 :

METHOD : reEnable
PURPOSE : This will re-enable the controls
PARMS :

RETURN : None

sndPanelMsg (amsgType : panelMsg) :

METHOD : sndPanelMsg
PURPOSE : This will display the appropriate message to the panel
PARMS :
amsgType [panelMsg] = the type of message that should be displayed
RETURN : None

cancelSLChart 0 :
PARMS : None

RETURN : None

cancelGroup 0 :

METHOD : cancelGroup
PURPOSE : This method will cancel an add to the record set for group
PARMS : None

RETURN : None

deleteSLGroup 0 :
PARMS : None

RETURN : None

deleteSLChart 0 :
PARMS : None

RETURN : None

saveSLChartRS 0 :
PARMS : None

RETURN : None

LOGICAL VIEW REPORT

moveGroupRs () :
PARMS : None

RETURN : None

checkForMissingData () : Boolean
PARMS : None

RETURN : Boolean

bindSLGroupFlds () :

METHOD : bindSLGroupFlds
PURPOSE : This method will bind the group text fields to the record set
PARMS : None

RETURN : None

newSLGroup () :
PARMS : None

RETURN : None

enableText () :

METHOD : enableText
PURPOSE : This method will enable the input text fields
PARMS : None

RETURN : None

setTextFields () :

METHOD : setTextFields
PURPOSE : This method will bind the text fields to the record sets
PARMS : None

RETURN : None

txtSLSearchGroup_KeyUp (KeyCode : Integer, Shift : Integer) :

METHOD : txtSLSearchGroup_KeyUp
PURPOSE : This method will determine if the input entered length is greater than 2
PARMS :
KeyCode [Integer] =
Shift [Integer] =
RETURN : None

LOGICAL VIEW REPORT

Form_QueryUnload (Cancel : Integer, UnloadMode : Integer) :
getSLChartOfAccounts (aSearch : groupID) :
PARMS : None
aSearch [groupID] = if a groupdid was supplied
RETURN : None

---

getSLGroupData 0 :

METHOD : getSLGroupData
PURPOSE : This method will retrieve the subledger Group data
PARMS : None

RETURN : None

---

bldSLCTranslation 0 :

METHOD : bldSLCTranslation
PURPOSE : This method will build the translation for the SLChartAccounts Grid
PARMS : None

RETURN : None

Form_Load 0 :
PARMS : None

RETURN : None

---

dgrdSLGroup_RowColChange (LastRow : Variant, LastCol : Integer) :
PARMS :
LastRow [Variant] =
LastCol [Integer] =
RETURN : None

---

dgrdSLGroup_PostEvent (MsgId : Integer) :
RETURN : None

---

dgrdSLGroup_MouseUp (Button : Integer, Shift : Integer, X : Single, Y : Single) :

METHOD : dgrdSLGroup_MouseUp
PURPOSE : This method will determine what options are available on the context menu
PARMS : None
Button [Integer] =
Shift [Integer] =
X [Single] =
Y [Single] =
RETURN : None

LOGICAL VIEW REPORT

---

dgrdSLGroup_BeforeRowColChange (Cancel : Integer) :

PARMS :
Cancel [Integer] =
RETURN : None

---

dgrdSLChartAccounts_MouseUp (Button : Integer, Shift : Integer, X : Single, Y : Single) :

METHOD : dgrdSLChartAccounts_MouseUp
PURPOSE : This method will determine what options are available on the context menu
PARMS : None
Button [Integer] =
Shift [Integer] =
X [Single] =
Y [Single] =
RETURN : None

---

dcboSlChart_Change 0 :
PARMS : None

RETURN : None

---

cmdSearch_Click 0 :
PARMS : None

RETURN : None

---

cmdClear_Click 0 :

METHOD : cmdClear_Click
PURPOSE : This method will clear the search field.
PARMS : None

RETURN : None

---

abarSubledger_Click (Tool : ActiveBarLibraryCtl.Tool) :

LOGICAL VIEW REPORT

modMain

Public Attributes:

HH_TP_HELP_WM_HELP : = &H11
HH_TP_HELP_CONTEXTMENU : = &H10
HH_HELP_CONTEXT : = &HF
HH_DISPLAY_TEXT_POPUP : = &HE
HH_GET_WIN_HANDLE : = &H6
HH_GET_WIN_TYPE : = &H5
HH_SET_WIN_TYPE : = &H4
HH_DISPLAY_TOPIC : = &H0
HELP_MAP_AE_WELCOME : Integer = 1
cBAD_ENTRY_BACKCOLOR : = vbYellow
cDISABLED_BACKCOLOR : = &H8000000F
cENABLED_BACKCOLOR : = &H80000005

Color Constants

UNLEN : = 256
GWW_HWNDPARENT : = (-8)

Public Operations:

activeBarLoad () :
disableActiveBar () :
RecordsetChanged (arsToBeChecked : ADOR.Recordset) : Boolean
pGetUserName () : String
htmlhelp (hwndCaller : Long, pszFile : String, uCommand : Long, dwData : Long) : Long

This Declare used for help window

SetWindowWord (hwnd : Long, nIndex : Long, wNewLong : Long) : Long

This declare used for floatable window (frmerrors)

GetUserName (lpBuffer : String, nSize : Long) : Long

-C52-

byval alJEID -
alProductID -
alBankID -
acurTXNAmount -
astrDRCRIND -
alCOAID -
astrPostPeriod -

Outputs: None
Returns: None

CreateSLMonthsBals(long byval SLBalanceID, currency byval acurTXNAmount, string byval astrPostPeriod)

Class: IPostSL
Description: None
Inputs: byval SLBalanceID -
byval acurTXNAmount -
byval astrPostPeriod -
Outputs: None
Returns: None

FinalizeProcessing()

Class: IPostSL
Description: None
Inputs: None
Outputs: None
Returns: None

WHAT IS CLAIMED IS:

1. A method of asset level accounting using a lease and loan sub-ledger accounting system (10), the accounting system including a lease and loan accounting engine (12), a plurality of sub-ledger accounting components independent from the accounting engine, and a plurality of programmatic interfaces (140) enabling communication with components of the accounting engine, said accounting system running within an operational system, said method comprising the steps of:

isolating accounting functions from the operational system; and

providing sub-ledger transaction detail.

2. A method according to Claim 1 further comprising the step of providing multi-national detail.

3. A method according to Claim 1 further comprising the step of internally and externally referring to financial entities.

4. A method according to Claim 1 further comprising the step of supporting multiple pricing models.

5. A method according to Claim 1 further comprising the step of defining and adding information needed to support specific accounting requirements.

6. A method according to Claim 1 further comprising the step of identifying every transaction in the accounting system (10) using an audit transaction component (62).

7. A method according to Claim 6 further comprising the step of relating every accounting transaction with a corresponding operational transaction using an operational system (60) enabled with an audit transaction component (62).

8. A method according to Claim 1 further comprising the step of deriving the correct accounting entry for a lease or loan accounting event using a flexible event driven process model (50).

9. A method according to Claim 1 further comprising the step of supporting multiple fiscal calendars (162).

10. A method according to Claim 1 further comprising the step of supporting multiple generally accepted accounting principles.

11. A method according to Claim 1 further comprising the step of defining user rules for determining a correct accounting entry based on existing information.

12. A method according to Claim 1 further comprising the step of defining calculation rules for supporting financial calculations needed to properly account for leases and loans in multiple, different organizations.

13. A method according to Claim 1 further comprising the step of specifying country, business, or product specific exceptions to an accounting event.

14. A method according to Claim 1 further comprising the step of defining financial asset grouping mechanisms.

15. A method according to Claim 1 further comprising the step of using a user definable financial asset grouping mechanism to summarize by groups of vendors, customers, branches, or offices.

16. A method according to Claim 1 further comprising the step of supporting account level accounting.

17. A lease and loan sub-ledger accounting system (10) for providing sub-ledger transaction detail for asset level accounting, said accounting system comprising

a lease and loan accounting engine (12);

a plurality of sub-ledger accounting components independent from said accounting engine; and

a plurality of programmatic interfaces enabling communication of said sub- ledger accounting components with said accounting engine.

18. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components provides multi-national detail.

19. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises internal and external references to financial entities.

20. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system supports multiple pricing models and multiple operational systems.

21. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises capability for a user to define and add information needed to support specific accounting requirements.

22. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises an audit transaction component (62) identifying every transaction in said accounting system.

23. A system (10) according to Claim 22 wherein said audit transaction component (62) allows an operational system to relate every accounting transaction with a corresponding operational transaction.

24. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises a flexible event driven process model (50) to allow an accounting system to derive a correct accounting entry for a lease or loan accounting event.

25. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises support for multiple fiscal calendars (162).

26. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises support for multiple generally accepted accounting principles.

27. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises user defined finance rules for determining a correct accounting entry based on existing information.

28. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises user defined calculation rules for supporting financial calculations needed to properly account for leases and loans in multiple, different business organizations.

29. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises event modifiers (102) specifying country, business, or product specific exceptions to an accounting event.

30. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises a user definable financial asset grouping mechanism .

31. A system (10) according to Claim 30 wherein said user definable financial asset grouping mechanism allows summarization by groups, said groups comprising vendors, customers, branches, or offices.

32. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises support for account level accounting.

33. A system (10) according to Claim 17 wherein at least one of said sub-ledger accounting components of said accounting system further comprises stream representations (100) of compressed data.

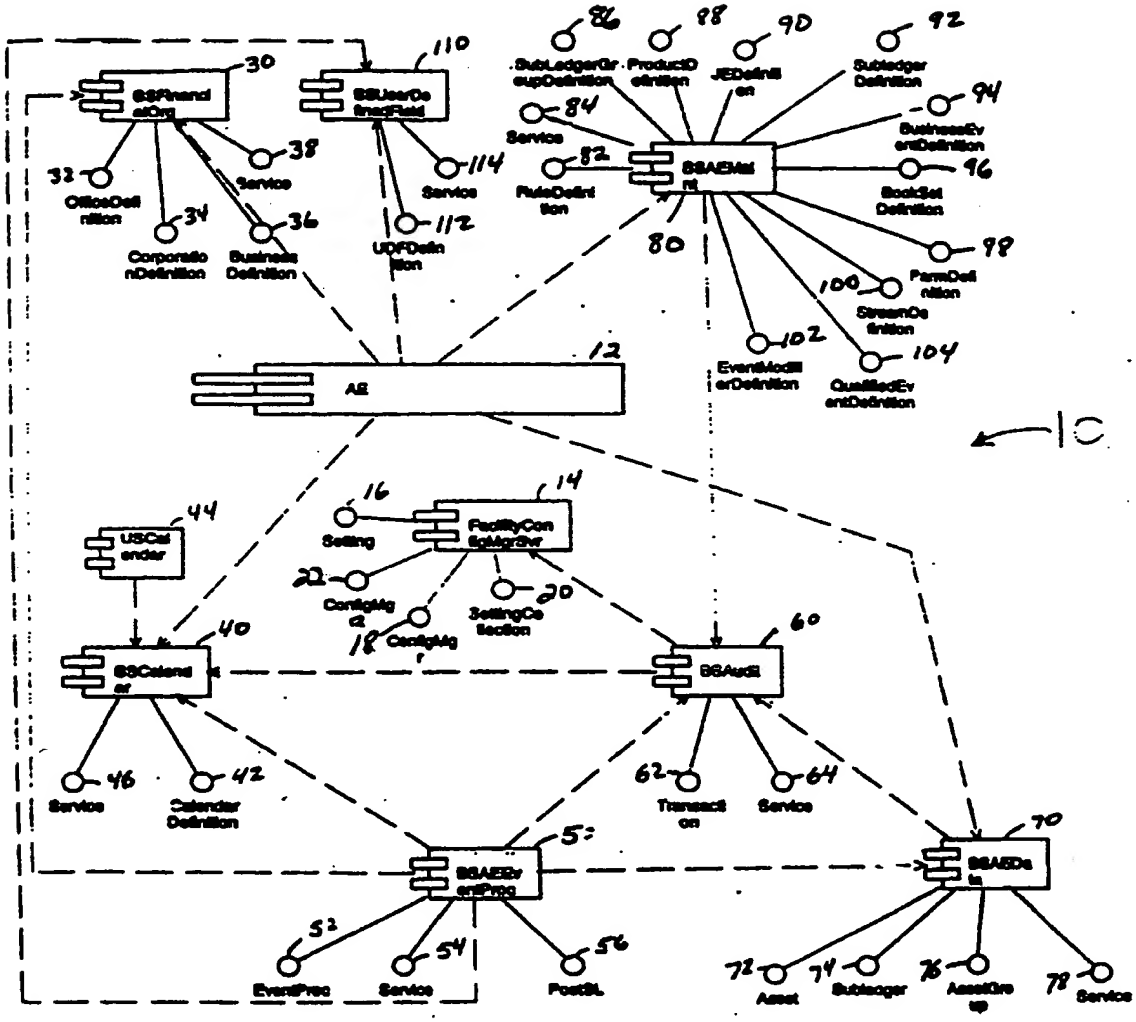

30
110
86
88
90
92
94
96
98
100
102
104
12
AE
10
16
14
44
40
60
46
42
62
64
50
70
52
54
56
72
74
76
78
80
82
84
112
114
Service
Setting
Calendar Definition
Transaction
Asset
Subledger
Service

FIG. 1

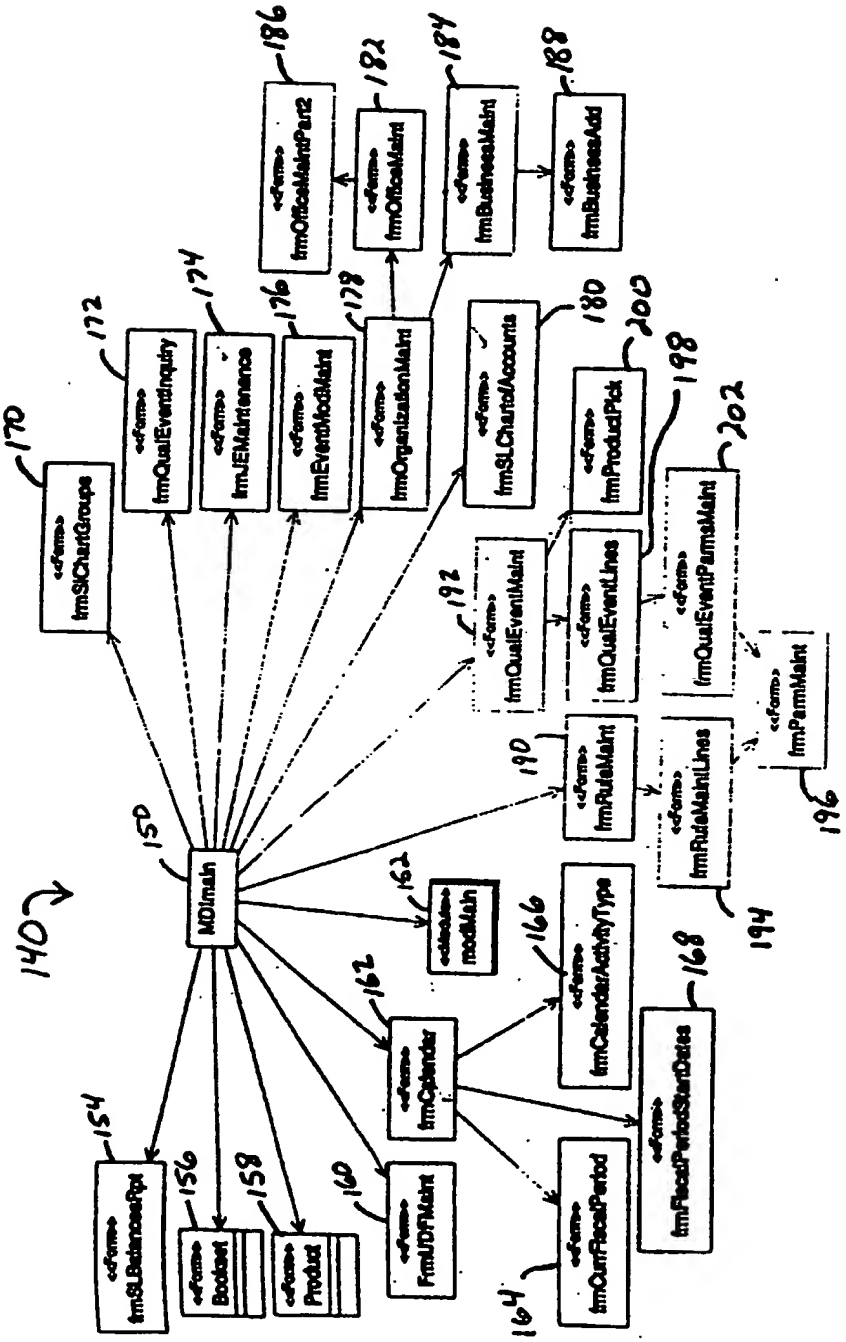

140
MDImain
150
frmSLChartGroups
170
frmQualEventInquiry
172
frmJEMaintenance
174
frmEventModMaint
176
frmOrganizationMaint
178
frmOfficeMaintPart2
186
frmOfficeMaint
182
frmBusinessMaint
184
frmBusinessAdd
188
frmSLChartofAccounts
180
frmProductPick
200
frmQualEventMaint
192
frmQualEventLines
198
frmQualEventParmsMaint
202
frmRuleMaint
190
frmRuleMaintLines
194
frmParmMaint
196
modMain
162
frmCalendar
162
frmCalendarActivityType
166
frmFiscalPeriodStartDates
168
frmOurFiscalPeriod
164
frmUDFMaint
160
Product
158
Bookset
156
frmSLBalanceRpt
154

FIG. 2